polysat version 1.3 Tutorial Manual

Lindsay V. Clark <lvclark@illinois.edu>
University of Illinois, Urbana-Champaign
Department of Crop Sciences
`http://openwetware.org/wiki/Polysat`

September 21, 2014

Contents

1 Introduction

The R package `polysat` provides 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, Structure, POPDIST, and STRand. It can also calculate pairwise genetic distances between samples, assist the user in estimating ploidy based on allele number, and estimate allele frequencies and F_{ST}. 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 `?save` or `?%in%`) and also get more detailed information on polysat functions (e.g. `?read.GeneMapper`).

The examples will be easiest to understand if you follow along with them and think about the purpose of each line of code. A file called "polysattuto-

rial.R" in the "doc" subdirectory of the package installation can be opened with a text editor and contains all of the R input found in this manual.

2 Obtaining and installing polysat

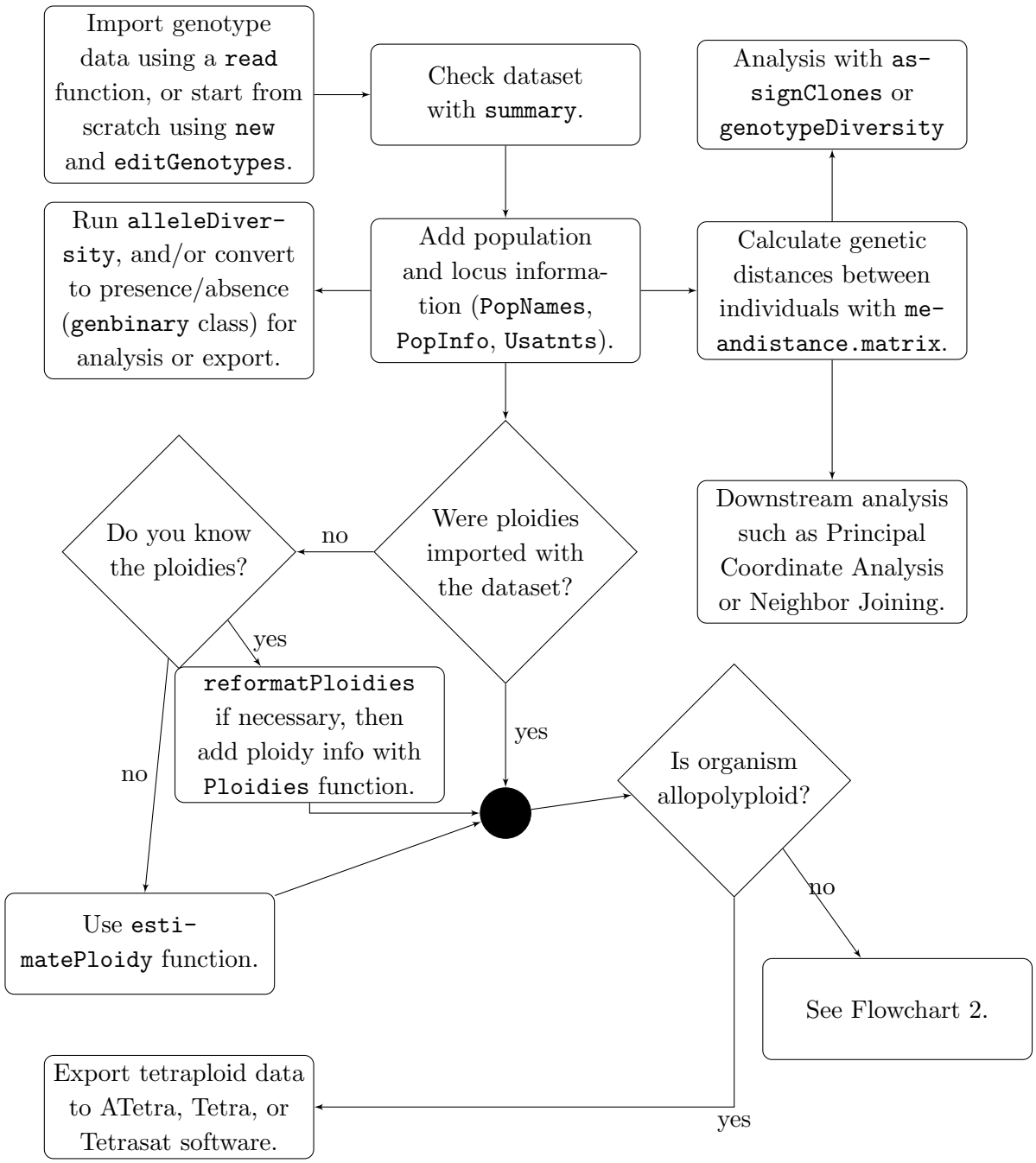
The R console and base system can be obtained at `http://www.r-project.org/`. Once R is installed, polysat can be installed and loaded by typing the following commands into the R console (note that combinat is only needed if the `Bruvo.distance` function is being used):

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

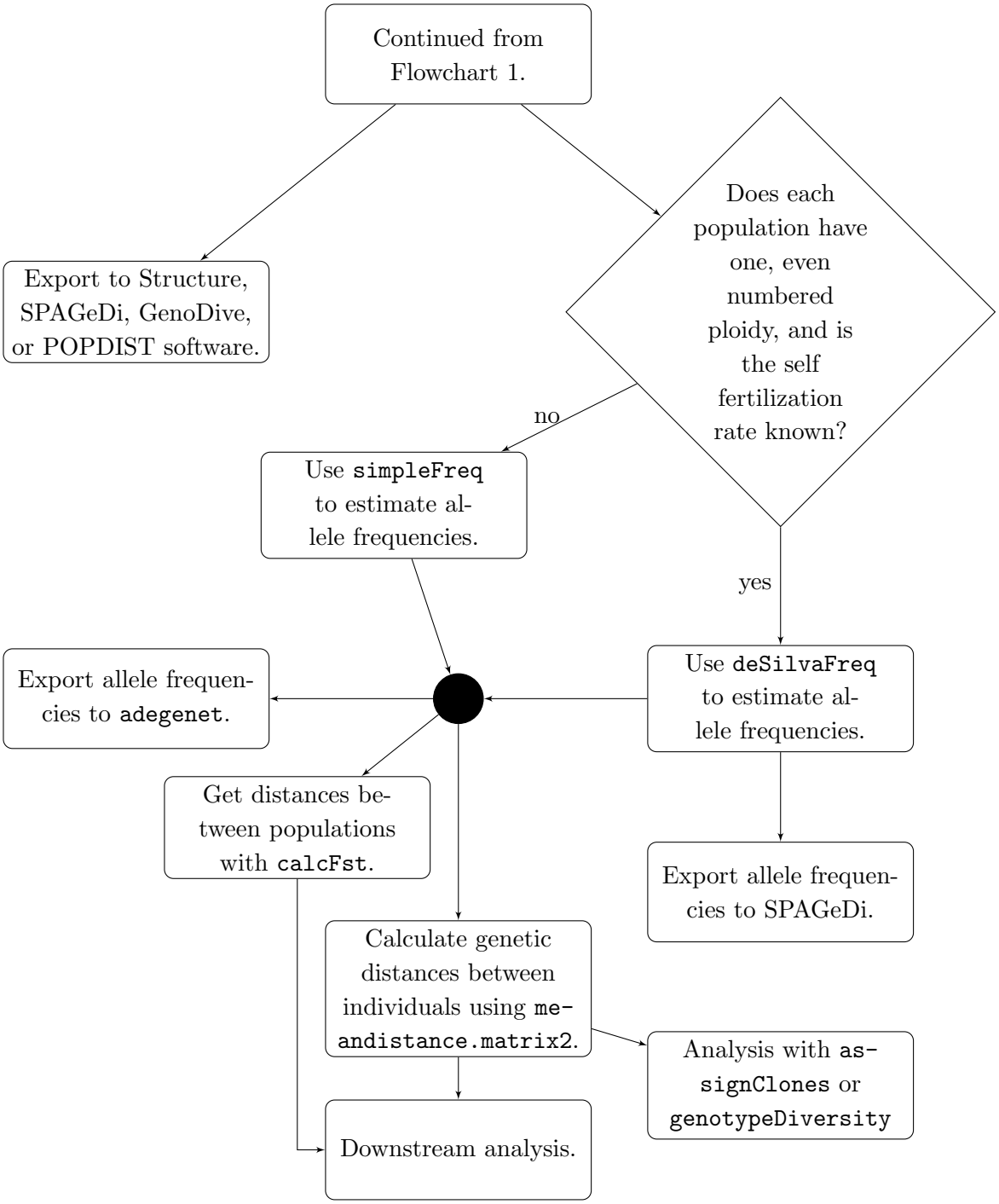
If you quit and restart R, you will not have to re-install the package but you will need to load it again (using the `library` function as shown above).

3 Workflow overview

The flowcharts on the next two pages give an overview of the steps required for the most common analyses performed in `polysat`. The first steps will always be importing or inputing the genotype data and making sure that the dataset contains information about populations and microsatellite repeat lengths. Different analysis and export functions are then available depending on whether the ploidy is known, whether the organism is autopolyploid or allopolyploid, and whether the selfing rate is known.



Flowchart 1. Functions for allopolyploid or autopolyploid data.



Flowchart 2. Functions for autopolyploid data only.

4 Getting Started: A Tutorial

4.1 Creating a dataset

As with any genetic software, the first thing you want to do is import your data. For this tutorial, go into the "extdata" directory of the polysat package installation, and find a file called "GeneMapperExample.txt". Open this file in a text editor and inspect its contents. This file contains simulated genotypes of 300 diploid and tetraploid individuals at three loci. Move this text file into the R working directory. The working directory can be changed with the `setwd` function, or identified with the `getwd` function:

```
> getwd()

[1] "/home/lvclark/RPackages/polysat/vignettes"
```

Then read the file using the `read.GeneMapper` function, and assign the dataset a name of your choice (`simgen` in this example) by typing:

```
> simgen <- read.GeneMapper("GeneMapperExample.txt")
```

The dataset now exists as an object in R. The following commands display, respectively, some basic information about the dataset, the sample and locus names, a subset of the genotypes, and a list of which genotypes are missing.

```
> summary(simgen)

Dataset with allele copy number ambiguity.
Insert dataset description here.
Number of missing genotypes: 5
300 samples, 3 loci.
1 populations.
Ploidies: NA
Length(s) of microsatellite repeats: NA

> Samples(simgen)
```

```
  [1] "A1"   "A2"   "A3"   "A4"   "A5"   "A6"   "A7"
  [8] "A8"   "A9"   "A10"  "A11"  "A12"  "A13"  "A14"
 [15] "A15"  "A16"  "A17"  "A18"  "A19"  "A20"  "A21"
 [22] "A22"  "A23"  "A24"  "A25"  "A26"  "A27"  "A28"
 [29] "A29"  "A30"  "A31"  "A32"  "A33"  "A34"  "A35"
 [36] "A36"  "A37"  "A38"  "A39"  "A40"  "A41"  "A42"
 [43] "A43"  "A44"  "A45"  "A46"  "A47"  "A48"  "A49"
 [50] "A50"  "A51"  "A52"  "A53"  "A54"  "A55"  "A56"
 [57] "A57"  "A58"  "A59"  "A60"  "A61"  "A62"  "A63"
 [64] "A64"  "A65"  "A66"  "A67"  "A68"  "A69"  "A70"
 [71] "A71"  "A72"  "A73"  "A74"  "A75"  "A76"  "A77"
 [78] "A78"  "A79"  "A80"  "A81"  "A82"  "A83"  "A84"
 [85] "A85"  "A86"  "A87"  "A88"  "A89"  "A90"  "A91"
 [92] "A92"  "A93"  "A94"  "A95"  "A96"  "A97"  "A98"
 [99] "A99"  "A100" "B1"   "B2"   "B3"   "B4"   "B5"
[106] "B6"   "B7"   "B8"   "B9"   "B10"  "B11"  "B12"
[113] "B13"  "B14"  "B15"  "B16"  "B17"  "B18"  "B19"
[120] "B20"  "B21"  "B22"  "B23"  "B24"  "B25"  "B26"
[127] "B27"  "B28"  "B29"  "B30"  "B31"  "B32"  "B33"
[134] "B34"  "B35"  "B36"  "B37"  "B38"  "B39"  "B40"
[141] "B41"  "B42"  "B43"  "B44"  "B45"  "B46"  "B47"
[148] "B48"  "B49"  "B50"  "B51"  "B52"  "B53"  "B54"
[155] "B55"  "B56"  "B57"  "B58"  "B59"  "B60"  "B61"
[162] "B62"  "B63"  "B64"  "B65"  "B66"  "B67"  "B68"
[169] "B69"  "B70"  "B71"  "B72"  "B73"  "B74"  "B75"
[176] "B76"  "B77"  "B78"  "B79"  "B80"  "B81"  "B82"
[183] "B83"  "B84"  "B85"  "B86"  "B87"  "B88"  "B89"
[190] "B90"  "B91"  "B92"  "B93"  "B94"  "B95"  "B96"
[197] "B97"  "B98"  "B99"  "B100" "C1"   "C2"   "C3"
[204] "C4"   "C5"   "C6"   "C7"   "C8"   "C9"   "C10"
[211] "C11"  "C12"  "C13"  "C14"  "C15"  "C16"  "C17"
[218] "C18"  "C19"  "C20"  "C21"  "C22"  "C23"  "C24"
[225] "C25"  "C26"  "C27"  "C28"  "C29"  "C30"  "C31"
[232] "C32"  "C33"  "C34"  "C35"  "C36"  "C37"  "C38"
[239] "C39"  "C40"  "C41"  "C42"  "C43"  "C44"  "C45"
[246] "C46"  "C47"  "C48"  "C49"  "C50"  "C51"  "C52"
```

```
[253] "C53"  "C54"  "C55"  "C56"  "C57"  "C58"  "C59"
[260] "C60"  "C61"  "C62"  "C63"  "C64"  "C65"  "C66"
[267] "C67"  "C68"  "C69"  "C70"  "C71"  "C72"  "C73"
[274] "C74"  "C75"  "C76"  "C77"  "C78"  "C79"  "C80"
[281] "C81"  "C82"  "C83"  "C84"  "C85"  "C86"  "C87"
[288] "C88"  "C89"  "C90"  "C91"  "C92"  "C93"  "C94"
[295] "C95"  "C96"  "C97"  "C98"  "C99"  "C100"
```

```
> Loci(simgen)

[1] "loc1" "loc2" "loc3"

> viewGenotypes(simgen, samples=paste("A", 1:20, sep=""), loci="loc1")

Sample        Locus        Alleles
A1        loc1        110        112        106
A2        loc1        114        106        118        110
A3        loc1        114        102        100        106
A4        loc1        110        102        106        100
A5        loc1        106        112
A6        loc1        100        110        106
A7        loc1        112        108
A8        loc1        102        106
A9        loc1        112
A10        loc1        102        106        110        112
A11        loc1        114        100        112
A12        loc1        106        118
A13        loc1        110        112
A14        loc1        100        112        106
A15        loc1        100        112        114
A16        loc1        112        102        100
A17        loc1        102        106
A18        loc1        102        106
A19        loc1        114        102        110        118
A20        loc1        106        100        108

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

```
  Locus Sample
1  loc1    B54
2  loc1    B80
3  loc2    B48
4  loc3    A42
5  loc3    C22
```

Additional information that isn't in "GeneMapperExample.txt" can be added directly to the dataset in R. The commands below add a description to the dataset, name three populations and assign 100 individuals to each, and indicate the length of the microsatellite repeats.

```
> Description(simgen) <- "Dataset for the tutorial"
> PopNames(simgen) <- c("PopA", "PopB", "PopC")
> PopInfo(simgen) <- rep(1:3, each = 100)
> Usatnts(simgen) <- c(2, 3, 2)
```

If you need help understanding what the `PopInfo` assignment means, type the following commands (results are hidden here for the sake of space):

```
> rep(1:3, each = 100)
> PopInfo(simgen)
```

Samples can now be retrieved by population. (Results hidden as above.)

```
> Samples(simgen, populations = "PopA")
```

The `Usatnts` assignment function above indicates that loc1 and loc3 have dinucleotide repeats, while loc2 has trinucleotide repeats. The alleles are recorded here in terms of fragment length in nucleotides. If the alleles were instead recorded in terms of repeat number, the `Usatnts` values should be 1. These repeat lengths can be examined by typing:

```
> Usatnts(simgen)

loc1 loc2 loc3
   2    3    2
```

To edit genotypes after importing the data:

```
> simgen <- editGenotypes(simgen, maxalleles = 4)

Edit the alleles, then close the data editor window.
```

You can also add ploidy information to the dataset. The `estimatePloidy` function allows you to add or edit the ploidy information, using a table that shows you the mean and maximum number of alleles per sample. The samples in this dataset should be diploid or tetraploid, although many of them may have fewer alleles. Therefore, in the data editor that is generated by the command below, you should change `new.ploidy` values to `2` if the sample has a maximum of one allele per locus, and to `4` if a sample has a maximum of three alleles per locus. See `?Ploidies` or page 22 for a different way to edit ploidy values if they are already known.

```
> simgen <- estimatePloidy(simgen)

Edit the new.ploidy values, then close the data editor window.
```

Take another look at the summary now that you have added this extra data.

```
> summary(simgen)

Dataset with allele copy number ambiguity.
Dataset for the tutorial
Number of missing genotypes: 5
300 samples, 3 loci.
3 populations.
Ploidies: 4 2
Length(s) of microsatellite repeats: 2 3
```

Now that you have your dataset completed, it is not a bad idea to save a copy of it. It will be automatically saved in your R workspace for use in subsequent R sessions. However, the `save` function creates a separate file containing a copy of the dataset (or any other R object), which can be useful as a backup against accidental changes or a copy to open on another computer. The file containing the dataset can be opened again at a later date using the `load` function.

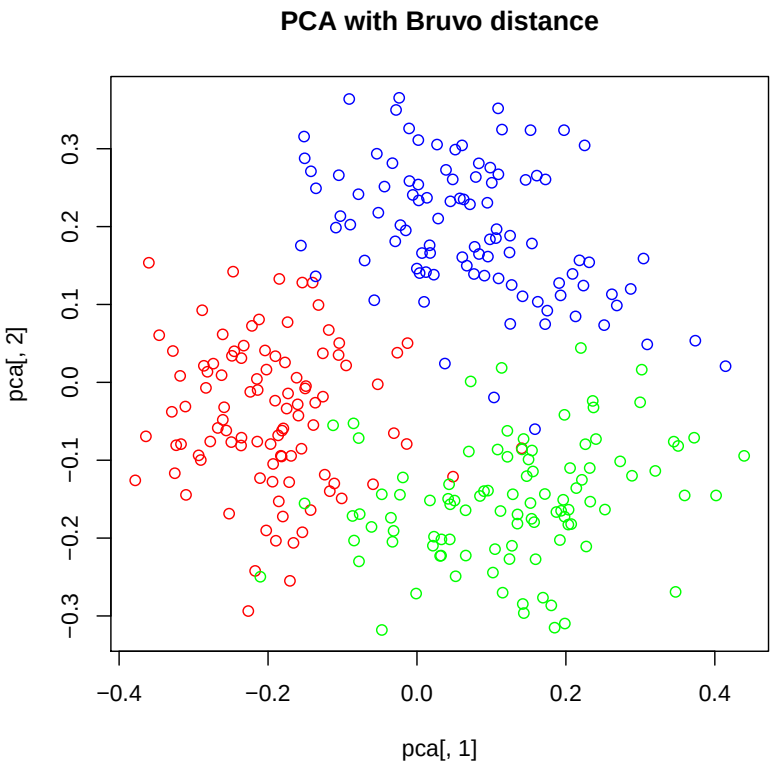
```
> save(simgen, file="simgen.RData")
```

4.2 Data analysis and export

4.2.1 Genetic distances between individuals

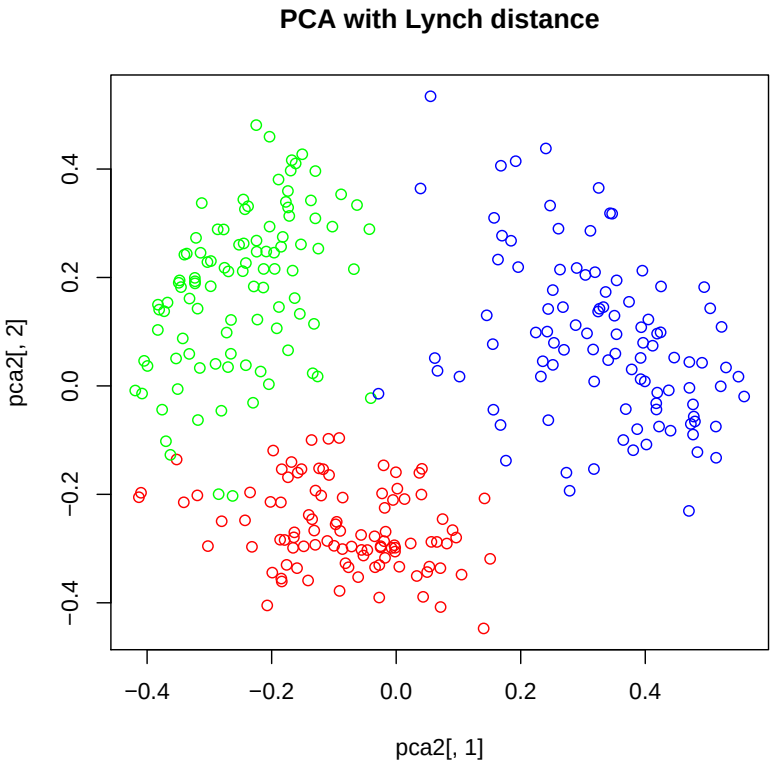
The code below calculates a pairwise distance matrix between all samples (using the default distance measure `Bruvo.distance`), performs Principal Coordinate Analysis (PCA) on the matrix, and plots the first two principal coordinates, with each population represented by a different color.

```
> testmat <- meandistance.matrix(simgen)

> pca <- cmdscale(testmat)
> mycol <- c("red", "green", "blue")
> plot(pca[,1], pca[,2], col=mycol[PopInfo(simgen)],
+      main = "PCA with Bruvo distance")
```



To conduct a PCA using the `Lynch.distance` measure, type:

```
> testmat2 <- meandistance.matrix(simgen, distmetric=Lynch.distance)

> pca2 <- cmdscale(testmat2)
> plot(pca2[,1], pca2[,2], col=rep(c("red", "green", "blue"), each=100),
+      main = "PCA with Lynch distance")
```



`Bruvo.distance` takes mutation into account, while `Lynch.distance` does not. (See `?Bruvo.distance`, `?Lynch.distance`, and section 6.3.) Since mutation was not part of the simulation that generated this dataset, the latter measure works better here for distinguishing populations.

If your data are autopolyploid, also see `?meandistance.matrix2` for another method of calculating inter-individual distances. If you have a mixed ploidy system in which the mechanism(s) for changes in ploidy are known, also see `?Bruvo2.distance`.

4.2.2 Working with subsets of the data

It is likely that you will want to perform some analyses on just a subset of your data. There are several ways to accomplish this in polysat. The `deleteSamples` and `deleteLoci` functions are designed to be fairly intuitive.

```
> simgen2 <- deleteSamples(simgen, c("B59", "C30"))
> simgen2 <- deleteLoci(simgen2, "loc2")
> summary(simgen2)

Dataset with allele copy number ambiguity.
Dataset for the tutorial
Number of missing genotypes: 4
298 samples, 2 loci.
3 populations.
Ploidies: 4 2
Length(s) of microsatellite repeats: 2
```

There are also a couple methods that involve using vectors of samples and loci that you *do* want to use. Let's make a vector of samples in populations A and B that are tetraploid, and then exclude a few samples that we don't want to analyze.

```
> samToUse <- Samples(simgen2, populations=c("PopA", "PopB"), ploidies=4)
> exclude <- c("A50", "A78", "B25", "B60", "B81")
> samToUse <- samToUse[!samToUse %in% exclude]
> samToUse

  [1] "A1"   "A2"   "A3"   "A4"   "A6"   "A10"  "A11"
  [8] "A14"  "A15"  "A16"  "A19"  "A20"  "A24"  "A26"
 [15] "A28"  "A29"  "A33"  "A34"  "A36"  "A37"  "A38"
 [22] "A39"  "A41"  "A42"  "A43"  "A46"  "A48"  "A49"
 [29] "A51"  "A57"  "A60"  "A61"  "A62"  "A63"  "A64"
 [36] "A66"  "A68"  "A69"  "A70"  "A76"  "A79"  "A81"
 [43] "A82"  "A83"  "A85"  "A86"  "A89"  "A90"  "A92"
 [50] "A94"  "A97"  "A98"  "A99"  "B2"   "B3"   "B5"
 [57] "B6"   "B10"  "B11"  "B12"  "B18"  "B19"  "B21"
 [64] "B22"  "B23"  "B24"  "B26"  "B28"  "B29"  "B31"
```

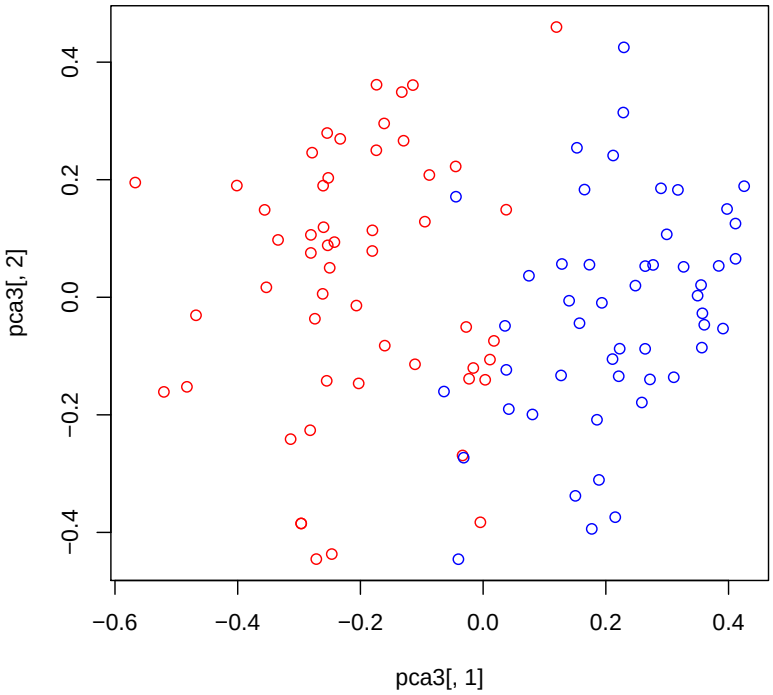
```
 [71] "B33"  "B37"  "B38"  "B40"  "B42"  "B43"  "B44"
 [78] "B45"  "B46"  "B47"  "B48"  "B51"  "B53"  "B55"
 [85] "B56"  "B63"  "B66"  "B67"  "B69"  "B70"  "B71"
 [92] "B75"  "B76"  "B78"  "B79"  "B83"  "B87"  "B88"
 [99] "B90"  "B91"  "B92"  "B95"  "B100"
```

You can subscript the dataset with square brackets, like you can with many other R objects. Note, however, that in this case you can't use square brackets to replace a subset of the dataset, just to access a subset of the dataset. A vector of samples should be placed first in the brackets, followed by a vector of loci.

```
> summary(simgen2[samToUse, "loc1"])

Dataset with allele copy number ambiguity.
Dataset for the tutorial
Number of missing genotypes: 0
103 samples, 1 loci.
2 populations.
Ploidies: 4
Length(s) of microsatellite repeats: 2
```

The analysis and data export functions all have optional `samples` and `loci` arguments where vectors of sample and locus names can indicate that only a subset of the data should be used.

```
> testmat3 <- meandistance.matrix(simgen2, samples = samToUse,
+                                 distmetric = Lynch.distance,
+                                 progress= FALSE)
> pca3 <- cmdscale(testmat3)
> plot(pca3[,1], pca3[,2], col=c("red", "blue")[PopInfo(simgen2)[samToUse]])
```



(If you are confused about how I got the color vector, I would encourage dissecting it: See what `PopInfo(simgen2)` gives you, what `PopInfo(simgen2)[samToUse]` gives you, and lastly what the result of `c("red", "blue")[PopInfo(simgen2)[samToUse]]` is.)

4.2.3 Population statistics

Allele frequencies are estimated in the example below. The example then uses these allele frequencies to calculate pairwise Wright's F_{ST} [14] values, first using all loci and then just two of the loci. See Section 6.4.1 for important information about allele frequency estimation.

```
> simfreq <- deSilvaFreq(simgen, self = 0.1, initNull = 0.01,
+                        samples = Samples(simgen, ploidies = 4))

Starting loc1
Starting loc1 PopA
```

```
64 repetitions for loc1 PopA
Starting loc1 PopB
106 repetitions for loc1 PopB
Starting loc1 PopC
84 repetitions for loc1 PopC
Starting loc2
Starting loc2 PopA
54 repetitions for loc2 PopA
Starting loc2 PopB
94 repetitions for loc2 PopB
Starting loc2 PopC
89 repetitions for loc2 PopC
Starting loc3
Starting loc3 PopA
104 repetitions for loc3 PopA
Starting loc3 PopB
117 repetitions for loc3 PopB
Starting loc3 PopC
105 repetitions for loc3 PopC
```

> simfreq

```
     Genomes  loc1.100   loc1.102   loc1.104  loc1.106
PopA     212 0.1202992 0.12041013 0.00000000 0.2196366
PopB     208 0.0000000 0.16964161 0.09127732 0.0666518
PopC     180 0.1546742 0.01733696 0.24074235 0.0000000
       loc1.108   loc1.110  loc1.112  loc1.114   loc1.116
PopA 0.03591695 0.14287772 0.1542292 0.1251016 0.00000000
PopB 0.00000000 0.12865007 0.0000000 0.1251792 0.09286717
PopC 0.10203928 0.03436444 0.1477607 0.0749076 0.18553453
       loc1.118  loc1.null   loc2.143   loc2.146  loc2.149
PopA 0.07118362 0.01034496 0.00000000 0.16292064 0.0000000
PopB 0.30132333 0.02440948 0.39112389 0.05846641 0.1964645
PopC 0.02862591 0.01401403 0.09199651 0.12284567 0.1100339
       loc2.152  loc2.155  loc2.158  loc2.161  loc2.164
PopA 0.01937013 0.2277736 0.2318032 0.2269041 0.1208905
PopB 0.00000000 0.0000000 0.1737714 0.1586404 0.0000000
```

```
PopC 0.30329792 0.1475359 0.0000000 0.0000000 0.2080345
      loc2.null   loc3.210   loc3.212  loc3.214   loc3.216
PopA 0.01033780 0.08777834 0.00000000 0.1171561 0.07825934
PopB 0.02153341 0.00000000 0.15664870 0.0000000 0.00000000
PopC 0.01625563 0.21567201 0.06139389 0.0000000 0.13814503
       loc3.218  loc3.220   loc3.222   loc3.224  loc3.226
PopA 0.27813128 0.0000000 0.15201002 0.00000000 0.0000000
PopB 0.37855398 0.0000000 0.15477761 0.15861852 0.0000000
PopC 0.09445973 0.1538148 0.06183346 0.08256635 0.1684937
       loc3.228   loc3.230  loc3.null
PopA 0.05675610 0.20737987 0.02252894
PopB 0.02972989 0.08606954 0.03560175
PopC 0.00000000 0.00000000 0.02362112
```

```
> simFst <- calcFst(simfreq)
> simFst
```

```
           PopA       PopB       PopC
PopA 0.00000000 0.05068795 0.05453103
PopB 0.05068795 0.00000000 0.07098261
PopC 0.05453103 0.07098261 0.00000000
```

```
> simFst12 <- calcFst(simfreq, loci=c("loc1", "loc2"))
> simFst12
```

```
           PopA       PopB       PopC
PopA 0.00000000 0.06004514 0.05597902
PopB 0.06004514 0.00000000 0.07356898
PopC 0.05597902 0.07356898 0.00000000
```

4.2.4 Genotype data export

Lastly, you may want to export your data for use in another program. Below is a simple example of data export for the software Structure. Additional export functions are described in sections 6.2 and 7.1. More details on the options for all of these functions are found in their respective help files.

In this example, both dipliod and tetraploid samples are included in the file. The `ploidy` argument indicates how many lines per individual the file should have.

```
> write.Structure(simgen, ploidy = 4, file="simgenStruct.txt")
```

5 How data are stored in polysat

In the tutorial above, you learned some ways of creating, viewing, and editing a dataset in polysat. This section goes into more details of the underlying data structure in polysat. This is particularly useful to understand if you want to extend the functionality of the package, but it may clear up some confusion for basic polysat users as well.

polysat uses the S4 class system in R. "Class" and "object" are two computer science terms that are introduced in Section 3 of *An Introduction to R*. Whenever you create a vector, data frame, matrix, list, etc. you are creating an object, and the class of the object defines which of these the object is. Furthermore, a class has certain "methods" defined for it so that the user can interact with the object in pre-specified ways. For example, if you use `mean` on a matrix, you will get the mean of all elements of the matrix, while if you use `mean` on a data frame, you will get the mean of each column; `mean` is a generic function with different methods for these two classes. S4 classes in R have "slots", where each slot can hold an object of a certain class. Methods define how the user can access, replace, and manipulate the data in these slots.

5.1 The "genambig" class

The object that you created with the `read.GeneMapper` function in the tutorial is of the class `"genambig"`. This class has the slots `Description` (a character string or character vector describing the dataset), `Genotypes` (a two-dimensional list of vectors, where each vector contains all unique alleles for a particular sample at a particular locus), `Missing` (the symbol for a missing genotype), `Usatnts` (a vector containing the repeat length of each locus, or 1 if alleles for that locus are already in terms of repeat number rather than nucleotides), `Ploidies` (an object of the class `"ploidysuper"`, which can contain a single value, a vector indexed by sample or locus, or a matrix indexed by sample and locus, any of which can contain integers to indicate ploidy), `PopNames` (the name of each population), and `PopInfo` (the population identity of each sample, using integers that correspond to the position of the population name in `PopNames`). You'll notice that there

aren't slots to hold sample or locus names, which are stored as the `names` and `dimnames` of the objects in the other slots.

```
> showClass("genambig")

Class "genambig" [package "polysat"]

Slots:

Name:    Genotypes Description     Missing     Usatnts
Class:       array   character         ANY     integer

Name:     Ploidies     PopInfo    PopNames
Class: ploidysuper     integer   character

Extends: "gendata"
```

To create a `"genambig"` object from scratch without using one of the data import functions, first create two character vectors to contain sample and locus names, respectively. These vectors are then used as arguments to the `new` function.

```
> mysamples <- c("indA", "indB", "indC", "indD", "indE", "indF")
> myloci <- c("loc1", "loc2", "loc3")
> mydataset <- new("genambig", samples=mysamples, loci=myloci)
```

An object has now been created with all of the appropriate slots named according to sample and locus names.

```
> mydataset

An object of class "genambig"
Slot "Genotypes":
     loc1 loc2 loc3
indA -9   -9   -9
indB -9   -9   -9
indC -9   -9   -9
indD -9   -9   -9
```

```
indE -9   -9   -9
indF -9   -9   -9

Slot "Description":
[1] "Insert dataset description here."

Slot "Missing":
[1] -9

Slot "Usatnts":
loc1 loc2 loc3
  NA   NA   NA

Slot "Ploidies":
An object of class "ploidymatrix"
Slot "pld":
     loc1 loc2 loc3
indA   NA   NA   NA
indB   NA   NA   NA
indC   NA   NA   NA
indD   NA   NA   NA
indE   NA   NA   NA
indF   NA   NA   NA


Slot "PopInfo":
indA indB indC indD indE indF
  NA   NA   NA   NA   NA   NA

Slot "PopNames":
character(0)
```

In the tutorial you used some of the accessor and replacement functions for the "genambig" class. You can see a full list of them by typing:

```
> ?Samples
```

(`Present` and `Absent` are just for the `"genbinary"` class. More on that later.) Let's use some of these functions to fill in and examine the dataset.

```
> Loci(mydataset)

[1] "loc1" "loc2" "loc3"

> Loci(mydataset) <- c("L1", "L2", "L3")
> Loci(mydataset)

[1] "L1" "L2" "L3"

> Samples(mydataset)

[1] "indA" "indB" "indC" "indD" "indE" "indF"

> Samples(mydataset)[3] <- "indC1"
> Samples(mydataset)

[1] "indA"  "indB"  "indC1" "indD"  "indE"  "indF"

> PopNames(mydataset) <- c("Yosemite", "Sequoia")
> PopInfo(mydataset) <- c(1,1,1,2,2,2)
> PopInfo(mydataset)

 indA  indB indC1  indD  indE  indF
    1     1     1     2     2     2

> PopNum(mydataset, "Yosemite")

[1] 1

> PopNum(mydataset, "Sequoia") <- 3
> PopNames(mydataset)

[1] "Yosemite" NA         "Sequoia"

> PopInfo(mydataset)
```

```
 indA  indB indC1  indD  indE  indF
    1     1     1     3     3     3
```

```
> Ploidies(mydataset) <- c(4,4,4,4,4,6)
> Ploidies(mydataset)
```

```
      L1 L2 L3
indA   4  4  4
indB   4  4  4
indC1  4  4  4
indD   4  4  4
indE   4  4  4
indF   6  6  6
```

```
> Ploidies(mydataset)["indC1",] <- 6
> Ploidies(mydataset)
```

```
      L1 L2 L3
indA   4  4  4
indB   4  4  4
indC1  6  6  6
indD   4  4  4
indE   4  4  4
indF   6  6  6
```

```
> Usatnts(mydataset) <- c(2,2,2)
> Usatnts(mydataset)
```

```
L1 L2 L3
 2  2  2
```

```
> Description(mydataset) <- "Tutorial, part 2."
> Description(mydataset)
```

```
[1] "Tutorial, part 2."
```

```
> Genotypes(mydataset, loci="L1") <- list(c(122, 124, 128), c(124,126),
+                       c(120,126,128,130), c(122,124,130), c(128,130,132),
+                       c(126,130))
> Genotype(mydataset, "indB", "L3") <- c(150, 154, 160)
> Genotypes(mydataset)

      L1        L2 L3
indA  Numeric,3 -9 -9
indB  Numeric,2 -9 Numeric,3
indC1 Numeric,4 -9 -9
indD  Numeric,3 -9 -9
indE  Numeric,3 -9 -9
indF  Numeric,2 -9 -9

> Genotype(mydataset, "indD", "L1")

[1] 122 124 130

> Missing(mydataset)

[1] -9

> Missing(mydataset) <- -1
> Genotypes(mydataset)

      L1        L2 L3
indA  Numeric,3 -1 -1
indB  Numeric,2 -1 Numeric,3
indC1 Numeric,4 -1 -1
indD  Numeric,3 -1 -1
indE  Numeric,3 -1 -1
indF  Numeric,2 -1 -1
```

If you know a little bit more about S4 classes, you know that you can access the slots directly using the `@` symbol, for example:

```
> mydataset@Genotypes
```

```
      L1        L2 L3
indA  Numeric,3 -1 -1
indB  Numeric,2 -1 Numeric,3
indC1 Numeric,4 -1 -1
indD  Numeric,3 -1 -1
indE  Numeric,3 -1 -1
indF  Numeric,2 -1 -1
```

```
> mydataset@Genotypes[["indB","L1"]]
```

```
[1] 124 126
```

However, I STRONGLY recommend against accessing the slots in this way in order to replace (edit) the data. The replacement functions are designed to prevent multiple types of errors that could happen if the user edited the slots directly.

In section 4.1 you were introduced to the `find.missing.gen` function. There is a related function called `isMissing` that may be more useful from a programming standpoint.

```
> isMissing(mydataset, "indA", "L2")
```

```
[1] TRUE
```

```
> isMissing(mydataset, "indA", "L1")
```

```
[1] FALSE
```

```
> isMissing(mydataset)
```

```
         L1   L2    L3
indA  FALSE TRUE  TRUE
indB  FALSE TRUE FALSE
indC1 FALSE TRUE  TRUE
indD  FALSE TRUE  TRUE
indE  FALSE TRUE  TRUE
indF  FALSE TRUE  TRUE
```

To add more samples or loci to your dataset, you can create a second `"genambig"` object and then use the `merge` function to join them.

```
> moredata <- new("genambig", samples=c("indG", "indH"), loci=Loci(mydataset))
> Usatnts(moredata) <- Usatnts(mydataset)
> Description(moredata) <- Description(mydataset)
> PopNames(moredata) <- "Kings Canyon"
> PopInfo(moredata) <- c(1,1)
> Ploidies(moredata) <- c(4,4)
> Missing(moredata) <- Missing(mydataset)
> Genotypes(moredata, loci="L1") <- list(c(126,130,136,138), c(124,126,128))
> mydataset2 <- merge(mydataset, moredata)
> mydataset2

An object of class "genambig"
Slot "Genotypes":
      L1        L2 L3
indA  Numeric,3 -1 -1
indB  Numeric,2 -1 Numeric,3
indC1 Numeric,4 -1 -1
indD  Numeric,3 -1 -1
indE  Numeric,3 -1 -1
indF  Numeric,2 -1 -1
indG  Numeric,4 -1 -1
indH  Numeric,3 -1 -1

Slot "Description":
[1] "Tutorial, part 2."

Slot "Missing":
[1] -1

Slot "Usatnts":
L1 L2 L3
 2  2  2

Slot "Ploidies":
An object of class "ploidymatrix"
Slot "pld":
      L1 L2 L3
```

```
indA   4  4  4
indB   4  4  4
indC1  6  6  6
indD   4  4  4
indE   4  4  4
indF   6  6  6
indG   4  4  4
indH   4  4  4


Slot "PopInfo":
 indA  indB indC1  indD  indE  indF  indG  indH
    1     1     1     3     3     3     4     4

Slot "PopNames":
[1] "Yosemite"     NA             "Sequoia"
[4] "Kings Canyon"
```

5.2 How ploidy data is stored: "ploidysuper" and subclasses

You may have noticed that in the above example, ploidy information was stored in a matrix, whereas in section 4.1 it was stored in a vector following the use of the `estimatePloidy` function. In fact, ploidy can be stored in four formats: a single value if the entire dataset has uniform ploidy, a vector indexed by sample if ploidy varies by sample, a vector indexed by locus if ploidy varies by locus (*e.g.* if the species is polyploid undergoing diploidization), or a matrix indexed by sample and locus (*e.g* if some of your loci are on sex chromosomes, or if some individuals are aneuploid). The object in the `Ploidies` slot of the dataset is one of four subclasses of the `"ploidysuper"` class (see table below), and this in turn has a slot called `pld` that contains the ploidy data. To make things simple from the user's perspective, the `Ploidies` accessor and replacement functions interact directly with this `pld` slot.

Class	Format	Use
ploidyone	unnamed vector of length one	uniform ploidy for entire dataset
ploidysample	vector indexed by sample	samples vary in ploidy
ploidylocus	vector indexed by locus	loci vary in copy number
ploidymatrix	matrix indexed by sample, then locus	different samples have different numbers of copies of different loci

Note that most analyses that use ploidy information assume completely random segregation of alleles. If you are going to specify ploidy as varying by locus, make sure that random segregation is actually the case for all loci. (See sections on working with autopolyploid vs. allopolyploid data.) For example, if a locus is present on two homeologous chromosome pairs, you may record the ploidy for that locus as being four. However, since these chromosomes do not pair with each other at meiosis, many of the analyses in polysat that utilize ploidy do not apply.

Many of the data import functions for polysat will detect the ploidies of genotypes and automatically create a `"genambig"` object with the simplest ploidy format possible. Additionally, when the `estimatePloidies` function is used, ploidy is automatically changed to being indexed by sample. However, the user may also want to manually switch formats, and the `reformatPloidies` function exists for this purpose.

```
> ploidyexample <- new("genambig")
> Samples(ploidyexample)

[1] "ind1" "ind2"

> Loci(ploidyexample)

[1] "loc1" "loc2"

> Ploidies(ploidyexample)

     loc1 loc2
ind1   NA   NA
ind2   NA   NA
```

```
> ploidyexample <- reformatPloidies(ploidyexample, output="locus")
> Ploidies(ploidyexample)

loc1 loc2
  NA   NA

> ploidyexample <- reformatPloidies(ploidyexample, output="sample")
> Ploidies(ploidyexample)

ind1 ind2
  NA   NA

> ploidyexample <- reformatPloidies(ploidyexample, output="one")
> Ploidies(ploidyexample)

[1] NA

> Ploidies(ploidyexample) <- 4
> ploidyexample <- reformatPloidies(ploidyexample, output="matrix")
> Ploidies(ploidyexample)

     loc1 loc2
ind1    4    4
ind2    4    4
```

See `?reformatPloidies` for more information on how to change formats when there is already data in the `Ploidies` slot.

Ploidy may be indexed using square brackets, like normal vectors and matrices:

```
> Ploidies(ploidyexample)["ind1", "loc1"]

[1] 4
```

However, for programming purposes, ploidy can also be indexed by passing `samples` and `loci` arguments to the `Ploidies` accessor function. This allows new functions to be robust to the ploidy format that is being used.

```
> Ploidies(ploidyexample, "ind1", "loc1")
```

```
     loc1
ind1    4
```

```
> ploidyexample <- reformatPloidies(ploidyexample, output="one")
> Ploidies(ploidyexample, "ind1", "loc1")
```

```
[1] 4
```

5.3 The "gendata" and "genbinary" classes

The "genambig" class is actually a subclass of another class called "gendata". The Description, PopInfo, PopNames, Ploidies, Missing, and Usatnts slots, and their access and replacement methods, are all defined for "gendata", and are inherited by "genambig". The "genambig" class adds the Genotypes slot and the methods for interacting with it.

A second subclass of "gendata" is "genbinary". This class also has a Genotypes slot, but formatted as a matrix indicating the presence and absence of alleles. (See ?genbinary-class for more details.) It also adds a slot called Present and one called Absent to indicate the symbols used to represent the presence or absence of the alleles, the same way the Missing slot holds the symbol used to indicate missing data. Like "genambig", "genbinary" inherits all of the slots from "gendata", as well as the methods for accessing them.

The code below creates a "genbinary" object using a conversion function, then demonstrates how the genotypes are stored differently and how the functions from "gendata" remain the same.

```
> simgenB <- genambig.to.genbinary(simgen)
> Genotypes(simgenB, samples=paste("A", 1:20, sep=""), loci="loc1")
```

```
    loc1.100 loc1.102 loc1.104 loc1.106 loc1.108 loc1.110
A1         0        0        0        1        0        1
A2         0        0        0        1        0        1
A3         1        1        0        1        0        0
A4         1        1        0        1        0        1
A5         0        0        0        1        0        0
A6         1        0        0        1        0        1
A7         0        0        0        0        1        0
```

```
A8          0          1          0          1          0          0
A9          0          0          0          0          0          0
A10         0          1          0          1          0          1
A11         1          0          0          0          0          0
A12         0          0          0          1          0          0
A13         0          0          0          0          0          1
A14         1          0          0          1          0          0
A15         1          0          0          0          0          0
A16         1          1          0          0          0          0
A17         0          1          0          1          0          0
A18         0          1          0          1          0          0
A19         0          1          0          0          0          1
A20         1          0          0          1          1          0
    loc1.112 loc1.114 loc1.116 loc1.118
A1          1          0          0          0
A2          0          1          0          1
A3          0          1          0          0
A4          0          0          0          0
A5          1          0          0          0
A6          0          0          0          0
A7          1          0          0          0
A8          0          0          0          0
A9          1          0          0          0
A10         1          0          0          0
A11         1          1          0          0
A12         0          0          0          1
A13         1          0          0          0
A14         1          0          0          0
A15         1          1          0          0
A16         1          0          0          0
A17         0          0          0          0
A18         0          0          0          0
A19         0          1          0          1
A20         0          0          0          0
> PopInfo(simgenB)[Samples(simgenB, ploidies=2)]
  A5   A7   A8   A9  A12  A13  A17  A18  A21  A22  A23  A25
```

```
  1    1    1    1    1    1    1    1    1    1    1    1
A27  A30  A31  A32  A35  A40  A44  A45  A47  A50  A52  A53
  1    1    1    1    1    1    1    1    1    1    1    1
A54  A55  A56  A58  A59  A65  A67  A71  A72  A73  A74  A75
  1    1    1    1    1    1    1    1    1    1    1    1
A77  A78  A80  A84  A87  A88  A91  A93  A95  A96 A100   B1
  1    1    1    1    1    1    1    1    1    1    1    2
 B4   B7   B8   B9  B13  B14  B15  B16  B17  B20  B25  B27
  2    2    2    2    2    2    2    2    2    2    2    2
B30  B32  B34  B35  B36  B39  B41  B49  B50  B52  B54  B57
  2    2    2    2    2    2    2    2    2    2    2    2
B58  B59  B61  B62  B64  B65  B68  B72  B73  B74  B77  B80
  2    2    2    2    2    2    2    2    2    2    2    2
B82  B84  B85  B86  B89  B93  B94  B96  B97  B98  B99   C1
  2    2    2    2    2    2    2    2    2    2    2    3
 C3   C4   C6   C7   C8  C10  C11  C14  C16  C17  C20  C21
  3    3    3    3    3    3    3    3    3    3    3    3
C23  C25  C27  C28  C31  C32  C36  C37  C38  C39  C40  C44
  3    3    3    3    3    3    3    3    3    3    3    3
C46  C47  C48  C50  C56  C57  C59  C61  C64  C67  C68  C71
  3    3    3    3    3    3    3    3    3    3    3    3
C74  C75  C76  C77  C79  C80  C82  C83  C84  C85  C86  C87
  3    3    3    3    3    3    3    3    3    3    3    3
C90  C92  C93  C95  C96  C98
  3    3    3    3    3    3
```

The "genbinary" class exists to facilitate the import and export of genotype data formatted in a binary presence/absence format, for example:

```
> write.table(Genotypes(simgenB), file="simBinaryData.txt")
```

The "genbinary" class is also used by polysat to make some of the allele frequency calculations easier. simpleFreq internally converts a "genambig" object to a "genbinary" object in order to tally allele counts in populations.

The class system in polysat is set up so that anyone can extend it to better suit their needs. There seem to be as many ways of formatting genotype data as their are population genetic software, and so a new subclass of "gendata"

could be created with genotypes formatted in a different way. A user could also create a subclass of "genambig", for example to hold GPS or phenotypic data in addition to the data already stored in a "genambig" object. (See ?setClass, ?setMethod, and [2].)

6 Functions for autopolyploid data

In order to properly utilize polysat (and other software for polyploid data) it is important to understand the inheritance mode in your system. In an autopolyploid, all homologous chromosomes are equally capable of pairing with each other at meiosis, and thus at a given microsatellite locus, gametes can receive any combination of alleles from the parent. The same is not true of allopolyploids. This affects the distribution of genotypes in the population, and as a result affects all aspects of population genetic analysis.

The functions described below are specifically for autopolyploid data. Their potential (or lack thereof) for use on allopolyploid data is described in the next section.

6.1 Data import

Four other population genetic programs that I am aware of can handle polyploid microsatellite data with allele copy number ambiguity under polysomic inheritance (autopolyploidy): Structure [5, 4, 15, 8], SPAGeDi [7], GenoDive [13] (http://www.bentleydrummer.nl/software/software/GenoDive.html), and POPDIST [6][18].

In the "extdata" directory of the polysat installation there are files called "structureExample.txt", "spagediExample.txt", "genodiveExample.txt", "POPDISTexample1.txt" and "POPDISTexample2.txt". To import these into "genambig" objects, first copy them into your working directory, then perform the assignments:

```
> GDdata <- read.GenoDive("genodiveExample.txt")
> Structdata <- read.Structure("structureExample.txt", ploidy = 8)
> Spagdata <- read.SPAGeDi("spagediExample.txt")
> PDdata <- read.POPDIST(c("POPDISTexample1.txt", "POPDISTexample2.txt"))
```

Use summary, viewGenotypes, and the accessor functions (section 5.1) to examine the contents of the three "genambig" objects that you have just

created. All four of these import functions take population information from the file and put it into the object. The Structure, SPAGeDi, and POPDIST files are coded in a way that indicates the ploidy of each individual, so this information is written to the `"genambig"` object as well.

The data import functions have some additional options for input and output, which are described in more detail in the help files. In particular, any extra columns can optionally be extracted from a Structure file, and the spatial coordinates can optionally be extracted from a SPAGeDi file. There are also several options for how ploidy should be interpreted from Structure files.

```
> ?read.Structure
> ?read.SPAGeDi
```

polysat also supports three genotype formats that work for either autopolyploids or allopolyploids, but do not contain any population, ploidy, or other information: GeneMapper, STRand, and binary presence/absence. The tutorial in the beginning of this manual uses `read.GeneMapper` to import data. The "GenaMapperExample.txt" file contains the minimum amount of information needed in order to be read by the function. Full "Genotypes Table" files as exported from ABI GeneMapper®can also be read by `read.GeneMapper`, and further, the function can take a vector of file names rather than a single file name if the data are spread across multiple files. There are three additional GeneMapper example files in the "doc" directory, which can be read into a `"genambig"` object in this way:

```
> GMdata <- read.GeneMapper(c("GeneMapperCBA15.txt",
+                             "GeneMapperCBA23.txt",
+                             "GeneMapperCBA28.txt"))
```

`read.STRand` takes a slightly modified version of the BTH format output by the allele-calling software STRand [19]. Since this format uses one row per individual, the modified format for `polysat` includes a column to contain population information.

```
> # view the format
> read.table("STRandExample.txt", sep="\t", header=TRUE)
```

```
  Pop Ind       GSSR28           GSSR76       GSSR90
1 NC1   a 134/140/154*     135/144/158*      201/211
2 NC1   b 140/154/172* 106/135/144/158* 205/211/213*
3 NC1   c      140/154 107/136/145/158*      199/207
4 VA1   a            0          145/149 199/207/211*
5 VA1   b          154          106/145      201/207
6 VA1   c      138/172     107/136/149* 207/211/213*
7 VA1   d 138/140/154*          135/158      207/213
```

```
> # import the data
> STRdata <- read.STRand("STRandExample.txt")
> Samples(STRdata)

[1] "NC1a" "NC1b" "NC1c" "VA1a" "VA1b" "VA1c" "VA1d"

> PopNames(STRdata)

[1] "NC1" "VA1"
```

A binary presence/absence matrix can be read into R using the base function `read.table`. Arguments to this function give options about how the file is delimited and whether it has headers and/or row labels. The example file in the "extdata" directory can be read in the following way:

```
> domdata <- read.table("dominantExample.txt", header=TRUE,
+                       sep="\t", row.names=1)
```

Examine the data frame produced, and notice in particular that the column names are formatted as the locus and allele separated by a period. After this data frame is converted to a matrix, it can be used to create a `"genbinary"` object.

```
> domdata

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

```
> domdata <- as.matrix(domdata)
> PAdata <- new("genbinary", samples=c("ind1", "ind2", "ind3"),
+               loci=c("ABC1", "ABC2"))
> Genotypes(PAdata) <- domdata
```

A few functions in polysat will work directly on a "genbinary" object, but for most functions you will want to convert to a "genambig" object. Addition of population and other information can be done either before or after the conversion.

```
> PopInfo(PAdata) <- c(1,1,2)
> PAdata <- genbinary.to.genambig(PAdata)
```

6.2 Data export

Autopolyploid data can also be exported in the same formats that are available for import, except STRand. Additionally, data can be exported to the R package adegenet's "genind" presence/absence format (see `?gendata.to.genind`).

The `write.Structure` function requires that an overall ploidy for the file be specified, to indicate how many rows per individual to write. Individuals with higher ploidy than the overall ploidy will have alleles randomly removed, and individuals with lower ploidy will have the missing data symbol inserted in the extra rows. Additional arguments give the options to specify extra columns to include, to omit or include population information, and to specify the missing data symbol. The row of missing data symbols that is automatically written underneath marker names is the RECESSIVEALLELES row in Structure, indicating that allele copy number is ambiguous.

`write.Structure` was used in the tutorial in section 4.2.4, but below is another example with some of the options changed (see `?write.Structure` for more information). Here, `myexcol` is an array of data to be written into extra columns in the file.

```
> myexcol <- array(c(rep(0:1, each=150), seq(0.1, 30, by=0.1)), dim=c(300,2),
+                  dimnames = list(Samples(simgen), c("PopFlag", "Something")))
> myexcol[1:10,]
```

```
    PopFlag Something
A1        0       0.1
```

```
A2        0       0.2
A3        0       0.3
A4        0       0.4
A5        0       0.5
A6        0       0.6
A7        0       0.7
A8        0       0.8
A9        0       0.9
A10       0       1.0
```

```
> write.Structure(simgen, ploidy=4, file="simgenStruct2.txt",
+                 writepopinfo = FALSE, extracols = myexcol,
+                 missingout = -1)
```

The `write.GenoDive` function is fairly straightforward, with the only option being whether to code alleles as two or three digits. All alleles are converted to repeat number, using the information contained in the `Usatnts` slot of the `"genambig"` object.

```
> write.GenoDive(simgen, file="simgenGD.txt")
```

`write.SPAGeDi` has options for the number of digits used to code alleles as well as the character (or lack thereof) used to separate alleles. Alleles are converted to repeat numbers as in `write.GenoDive`. Additionally, a data frame of spatial coordinates can be supplied to the function to be written to the file. By default, the function will create two dummy columns for spatial coordinates, which the user can then fill in using a text editor or spreadsheet software. (See `?write.SPAGeDi`)

```
> write.SPAGeDi(simgen, file="simgenSpag.txt")
```

If you are using SPAGeDi to calculate relationship and kinship coefficients, also see the function `write.freq.SPAGeDi` for exporting allele frequencies from polysat to SPAGeDi for use in these calculations.

The `write.POPDIST` function does not have any options for formatting. In the example below, the `samples` argument is used to ensure that each population has uniform ploidy, which is a requirement of the POPDIST software.

```
> write.POPDIST(simgen, samples = Samples(simgen, ploidies=4),
+               file = "simgenPOPDIST.txt")
```

`write.GeneMapper` is very straightforward, without any special formatting options. This function was used to create the "GeneMapperExample.txt" file that is provided with the package. I do not know of any other software that will read the GeneMapper format, but it may be a convenient way for the user to store and edit genotypes.

```
> write.GeneMapper(simgen, file="simgenGM.txt")
```

To export a table of genotypes in binary presence/absence format, first convert the `"genambig"` object to a `"genbinary"` object, then write the `Genotypes` slot to a text file, adjusting the options of `write.table` to suit your needs. (See `?write.table`.)

```
> simgenPA <- genambig.to.genbinary(simgen)
> write.table(Genotypes(simgenPA), file="simgenPA.txt", quote=FALSE,
+             sep = ",")
```

6.3 Individual-level statistics

6.3.1 Estimating and exporting ploidies

The `estimatePloidy` function, which was demonstrated in section 4.1, is equally appropriate for autopolyploid and allopolyploid data. If you want to export the ploidy data, one method is the following:

```
> write.table(data.frame(Ploidies(simgen), row.names=Samples(simgen)),
+             file="simgenPloidies.txt")
```

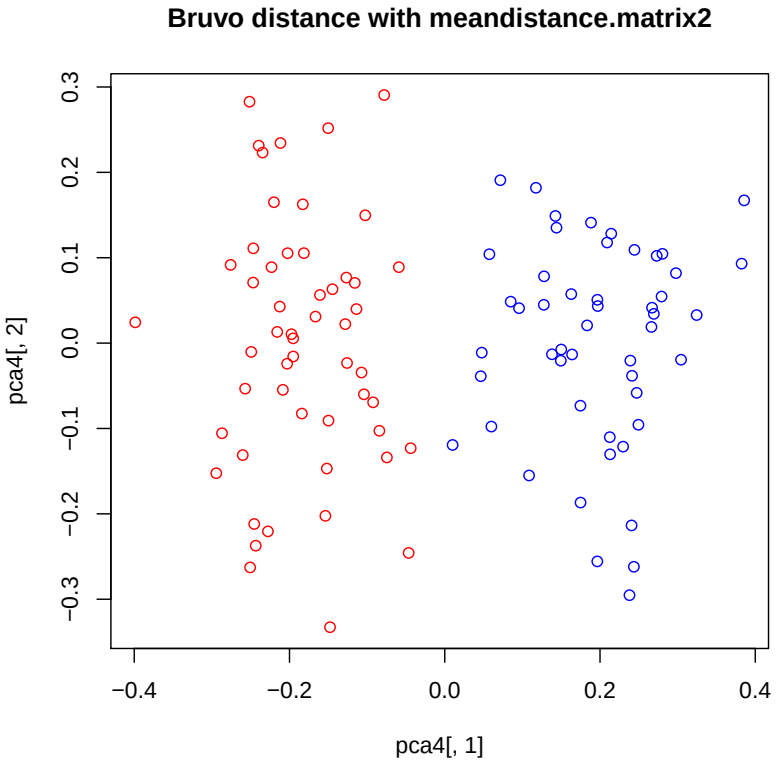
6.3.2 Inter-individual distances

A matrix of pairwise distances between individuals can be generated using the `meandistance.matrix` function, which was demonstrated in section 4.2.1. The most important argument is `distmetric`, or the distance measure that is used. The three options that are provided with polysat are `Bruvo.distance` and `Bruvo2.distance`, which take mutational distance between alleles into account [1], and `Lynch.distance`, which is a simple band-sharing measure

[11]. (The user can create functions to serve as additional distance measures, as long as the arguments are the same as those for `Bruvo.distance` and `Lynch.distance`.) The `progress` argument can be set to `TRUE` or `FALSE` to indicate whether the progress of the computation should be printed to the screen. The `all.distances` argument can also be set to `TRUE` or `FALSE` to indicate whether, in addition to the mean distance matrix, a three-dimensional array of distances by locus should be returned. There is also a `maxl` argument to indicate the threshold for `Bruvo.distance` to skip calculations that are too computationally intensive (see `?Bruvo.distance`). The function `Bruvo2.distance` has two additional arguments called `add` and `loss`, which when set to `TRUE` indicate that the models of genome addition and/or genome loss should be used, respectively.

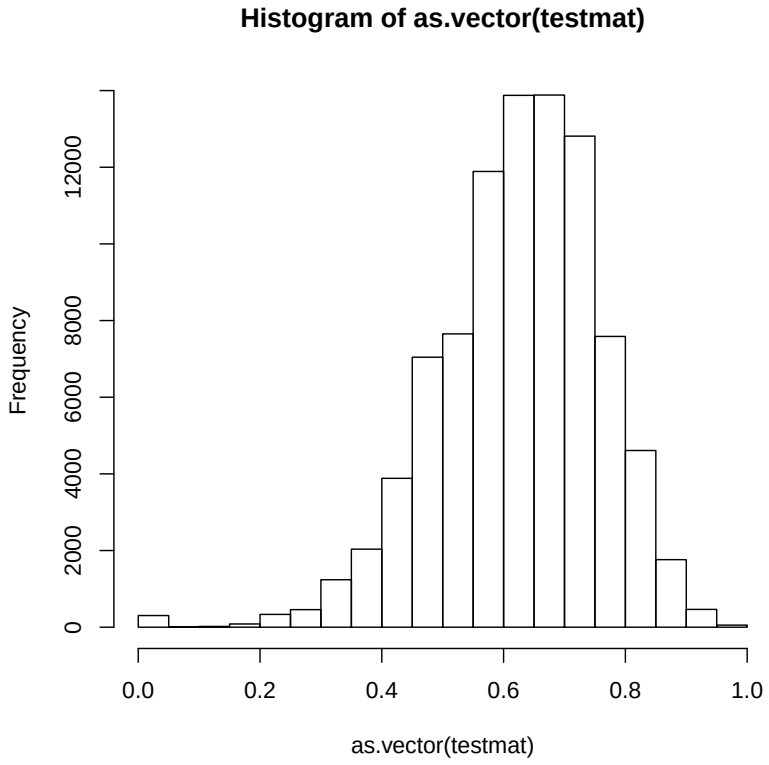
A second means of calculating inter-individual distances was introduced in polysat 1.2 and is called `meandistance.matrix2`. Whereas `meandistance.matrix` passes genotypes directly to `distmetric` with each allele present in only one copy, `meandistance.matrix2` uses ploidy, selfing rate, and allele frequencies to calculate the probabilities that a given ambiguous genotype represents any possible unambiguous genotype. Unambiguous genotypes are then passed to `distmetric`. The distance is a weighted average across all possible combinations of unambiguous genotypes. There is no advantage to using `Lynch.distance` with this function, but it may give improved results for `Bruvo.distance`, `Bruvo2.distance`, or a user-defined distance measure.

```
> testmat4 <- meandistance.matrix2(simgen, samples=samToUse, freq=simfreq,
+                                  self=0.2)

> pca4 <- cmdscale(testmat4)
> plot(pca4[,1], pca4[,2], col=c("red", "blue")[PopInfo(simgen)[samToUse]],
+      main="Bruvo distance with meandistance.matrix2")
```

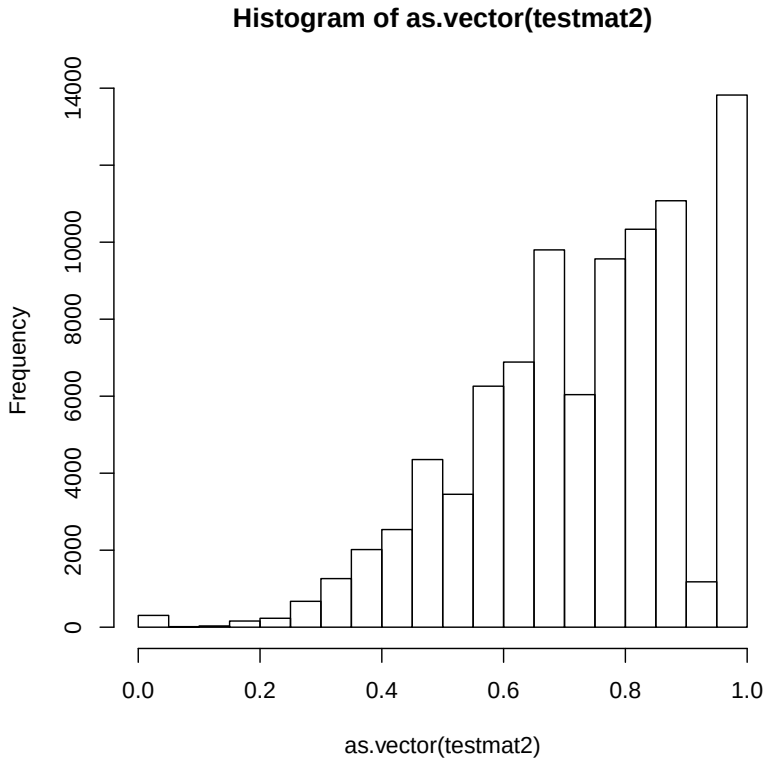


Besides the `cmdscale` function for performing Principal Coordinate Analysis on the resulting matrix, you may want to create a histogram to view the distribution of distances, or you may want to export the distance matrix for use in other software.

```
> hist(as.vector(testmat))
```




```
> hist(as.vector(testmat2))
```




```
> write.table(testmat2, file="simgenDistMat.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.

The following example first creates a vector indicating the subset of samples to use, both to save on computation time for the example and because missing data can be a problem for Principal Coordinate Analysis if fewer than three loci are used. An array of distances is then calculated, followed by the mean distance matrix for each combination of two loci.

```
> subsamples <- Samples(simgen, populations=1)
> subsamples <- subsamples[!isMissing(simgen, subsamples, "loc1") &
+                          !isMissing(simgen, subsamples, "loc2") &
+                          !isMissing(simgen, subsamples, "loc3")]
> Larray <- meandistance.matrix(simgen, samples=subsamples,
+                               progress=FALSE,
+                  distmetric=Lynch.distance, all.distances=TRUE)[[1]]
> mdist1.2 <- meandist.from.array(Larray, loci=c("loc1","loc2"))
> mdist2.3 <- meandist.from.array(Larray, loci=c("loc2","loc3"))
> mdist1.3 <- meandist.from.array(Larray, loci=c("loc1","loc3"))
```

As before, you can use `cmdscale` to perform Principal Coordinate Analysis and `plot` to visualize the results. Differences between plots reflect the effects of excluding loci.

6.3.3 Determining groups of asexually-related samples

Very similarly to the software GenoType [13], polysat can use a matrix of inter-individual distances to assign samples to groups of asexually-related individuals. This analysis can be performed on any matrix of distances calculated with `meandistance.matrix`, `meandistance.matrix2`, or a user-defined function that produces matrices in the same format. As in GenoType, a histogram such as those produced above may be useful for determining a distance threshold for distinguishing sexually- and asexually-related pairs of individuals. The data in `simgen` were simulated in a sexually-reproducing population, but let's pretend for the moment that there was some asexual reproduction, and we saw a bimodal distribution of distances with a cutoff of 0.2 between modes.

```
> clones <- assignClones(testmat, samples=paste("A", 1:100, sep=""),
+                        threshold=0.2)
> clones

  A1   A2   A3   A4   A5   A6   A7   A8   A9  A10  A11  A12
   1    2    3    4    5    6    7    8    9   10   11   12
 A13  A14  A15  A16  A17  A18  A19  A20  A21  A22  A23  A24
  13   14   15   16   17   18   19   20   21   22   23   24
 A25  A26  A27  A28  A29  A30  A31  A32  A33  A34  A35  A36
```

```
 25   26   27   28   15   29   30   31   32   33   34   35
A37  A38  A39  A40  A41  A42  A43  A44  A45  A46  A47  A48
 36   37   38   39   40   37   36    7   21   41   42   43
A49  A50  A51  A52  A53  A54  A55  A56  A57  A58  A59  A60
 44   45   46   47   48   49   27   50   14   51   17   28
A61  A62  A63  A64  A65  A66  A67  A68  A69  A70  A71  A72
 52   53   36   54   55   56   31   57   58   59   60   22
A73  A74  A75  A76  A77  A78  A79  A80  A81  A82  A83  A84
 61   22   62   63   64   65   66   67   68   69    3   55
A85  A86  A87  A88  A89  A90  A91  A92  A93  A94  A95  A96
 70   36   71   72   35   24   73   26   74   75   76   77
A97  A98  A99 A100
 78   37   79   80
```

Some of the individuals with similar genotypes have been assigned to the same clonal group.

Diversity statistics based on genotype frequencies are also available; see section 6.4.2.

6.4 Population statistics

6.4.1 Allele diversity and frequencies

Allele diversity, *i.e.* the number of alleles found at each locus, is easily calculated in polysat.

```
> simal <- alleleDiversity(simgen)
> simal$counts

        loc1 loc2 loc3
PopA       8    6    7
PopB       7    5    6
PopC       9    6    8
overall   10    8   11

> simal$alleles[["PopA","loc1"]]

[1] 100 102 106 108 110 112 114 118
```

There are two functions in polysat for estimating allele frequencies. If all of your individuals are the same, even-numbered ploidy and if you have a reasonable estimate of the selfing rate in your system, `deSilvaFreq` will give the most accurate estimate. For mixed ploidy systems, the `simpleFreq` function is available, but will be biased toward underestimating common allele frequencies and overestimating rare allele frequencies, which will cause an underestimation of F_{ST}. `deSilvaFreq` uses an iterative algorithm to estimate genotype frequencies based on allele frequencies and "allelic phenotype" frequencies, then recalculate allele frequencies from genotype frequencies [3]. `simpleFreq` simply assumes that in a partially heterozygous genotype, all alleles have an equal chance of being present in more than one copy.

Both allele frequency estimators take as the first argument a `"genambig"` or `"genbinary"` object, which must have the `PopInfo` and `Ploidies` slots filled in. The `self` argument for supplying the selfing rate is only applicable for `deSilvaFreq`. (See `?deSilvaFreq` for some other arguments that can be adjusted.) Both functions produce a data frame of allele frequencies, with populations in rows and alleles in columns. `deSilvaFreq` adds a null allele for each locus, while `simpleFreq` does not. In both cases the data frame will also have a column indicating the population size in number of genomes (*e.g.* four hexaploid individuals = 24 genomes).

The function `calcFst` takes the data frame produced by either allele frequency estimation, and produces a matrix containing pairwise F_{ST} values according to the original calculation by Wright [14]. Population sizes are weighted by number of genomes, rather than number of individuals.

Continuing the example from section 4.2.3, and comparing the results of `deSilvaFreq` and `simpleFreq`:

```
> simFst

           PopA       PopB       PopC
PopA 0.00000000 0.05068795 0.05453103
PopB 0.05068795 0.00000000 0.07098261
PopC 0.05453103 0.07098261 0.00000000

> simfreqSimple <- simpleFreq(simgen, samples = Samples(simgen, ploidies=4))
> simFstSimple <- calcFst(simfreqSimple)
> simFstSimple
```

```
            PopA       PopB           PopC
PopA 0.00000000 0.04738346  5.088305e-02
PopB 0.04738346 0.00000000  6.492718e-02
PopC 0.05088305 0.06492718 -1.323838e-16
```

Average allele frequencies can also be used by SPAGeDi for the calculation of relationship and kinship coefficients. SPAGeDi v1.3 can estimate allele frequencies using the same method as `simpleFreq`. However, if your data are appropriate for allele frequency estimation using `deSilvaFreq`, exporting the estimated allele frequencies to SPAGeDi should improve the accuracy of the relationship and kinship calculations. The `write.freq.SPAGeDi` function creates a file of allele frequencies in the format that is read by SPAGeDi.

```
> write.freq.SPAGeDi(simfreq, usatnts=Usatnts(simgen), file="SPAGfreq.txt")
```

The R package `adegenet`[9] can perform a number of calculations from allele frequencies, including five inter-population distance measures as well as Correspondance Analysis. The allele frequency tables produced by `polysat` can be converted to a format that can be read by `adegenet`.

```
> gpsimfreq <- freq.to.genpop(simfreq)
```

The object `gpsimfreq` that you just created can now be passed to the function `genpop` as the `tab` argument. See `?freq.to.genpop` for example code.

6.4.2 Genotype frequencies

For asexual organisms, you many want to calculate statistics based on the frequencies of genotypes in your populations. Two popular statistics for this, the Shannon index [16] and Simpson index [17], are provided with polysat. The function `genotypeDiversity` calculates either of these statistics or any user-defined statistic that can be calculated from a vector of counts. `genotypeDiversity` uses the function `assignClones` internally, so the same `threshold` argument may be set to allow for mutation or scoring error, or to group individuals by a larger distance threshold. This function examines loci individiually as well as the mean distance across all loci. Where ordinary allelic diversity statistics are not available due to allele copy number ambiguity, genotype diversity statistics for individual loci may be useful.

```
> testmat5 <- meandistance.matrix(simgen, all.distances=TRUE)

> simdiv <- genotypeDiversity(simgen, d=testmat5, threshold=0.2, index=Shannon)
> simdiv

         loc1     loc2     loc3  overall
PopA 3.304798 2.654090 3.379927 4.303583
PopB 3.269491 2.582981 2.913112 4.002176
PopC 3.373131 3.009567 3.415639 4.521993

> simdiv2 <- genotypeDiversity(simgen, d=testmat5, threshold=0.2, index=Simpson)
> simdiv2

           loc1       loc2       loc3     overall
PopA 0.03737374 0.08303030 0.03360132 0.005050505
PopB 0.03702924 0.08884766 0.05696970 0.020000000
PopC 0.03272727 0.05373737 0.03112760 0.001212121
```

The variance of the Simpson index may also be calculated, enabling the calculation of upper and lower bounds for a 95% confidence interval.

```
> simdiv2var <- genotypeDiversity(simgen, d=testmat5, threshold=0.2,
+                                 index=Simpson.var)
> simdiv2 - 2*simdiv2var^0.5

           loc1       loc2       loc3        overall
PopA 0.02438289 0.05915117 0.02113944  0.0004028381
PopB 0.02295020 0.06115142 0.04216782  0.0049848529
PopC 0.02133522 0.03403932 0.02007475 -0.0020469696

> simdiv2 + 2*simdiv2var^0.5

           loc1       loc2       loc3     overall
PopA 0.05036458 0.10690944 0.04606320 0.009698172
PopB 0.05110829 0.11654391 0.07177158 0.035015147
PopC 0.04411932 0.07343543 0.04218045 0.004471212
```

7 Functions for allopolyploid data

In order to properly analyze microsatellites as codominant markers in allopolyploids, knowledge is required about which alleles belong to which genome. In an autopolyploid, all alleles for a given marker will segregate according to Mendelian laws. In an allopolyploid, a microsatellite marker represents two or more loci that are behaving in a Mendelian fashion, but if treated as one locus will not appear to behave according to random segregation. For example, an autotetraploid with the genotype ABCD that self fertilizes can produce offspring with the genotype AABB. An allotetraploid with the same four alleles, but distributed as AB and CD across two genomes, cannot self to produce an AABB individual as both of these alleles come from one genome.

If you have knowledge from other analyses about which alleles belong to which genomes, when importing your data you can code each microsatellite marker as multiple loci. As long as each "locus" in the `"genambig"` object is behaving according to random segregation, the analysis and export functions for autopolyploid data described in the previous section are appropriate.

Otherwise, the following functionality is available for allopolyploids in polysat:

7.1 Data import and export

Data can be formatted for the software Tetrasat [12], Tetra [10], and ATetra [20] using polysat. These programs are intended to be robust to lack of knowledge of inheritance patterns of alleles in allotetraploids and will estimate allele frequencies and other statistics. See the help files for `write.Tetrasat` and `write.ATetra`.

`read.Tetrasat` (which produces a format readable by both Tetrasat and Tetra) and `read.ATetra` both take, as their only argument, the file name to be read. To import data from the example files "ATetraExample.txt" and "tetrasatExample.txt", use the commands:

```
> ATdata <- read.ATetra("ATetraExample.txt")
> Tetdata <- read.Tetrasat("tetrasatExample.txt")
```

The functions for writing these two file formats only require a `"genambig"` object and a file name. `Ploidies` and `PopInfo` are required in the object for both functions. `write.Tetrasat` additionally requires information in

the `Usatnts` slot. Since ATetra does not allow missing data, any missing genotypes that are encountered by `write.ATetra` are written to the console.

```
> write.ATetra(simgen, samples=Samples(simgen, ploidies=4), file="simgenAT.txt")

Missing data: B48 loc2
Missing data: A42 loc3
Missing data: C22 loc3

> write.Tetrasat(simgen, samples=Samples(simgen, ploidies=4),
+                file="simgenTet.txt")
```

Data for allopolyploids can also be imported and exported in GeneMapper, STRand, adegenet genind, and binary presence/absence formats, as described in the sections 6.1 and 6.2.

7.2 Individual-level and population statistics

The `Bruvo.distance` measure of inter-individual distances is best suited to autopolyploids but may work for allopolyploids under a special case. `Bruvo.distance` measures distances between all alleles at a locus for the two individuals being compared, under the premise that these alleles could be closely related to each other by mutation. If two alleles belong to two different allopolyploid genomes, it is not possible for them to be be closely related to each other even if their sizes are similar, since they are derived from different ancestral species. In the case where no allele from one allopolyploid genome is within three or four mutation steps of any allele from the other genome, it is possible for the value produced by `Bruvo.distance` to accurately reflect the genetic similarity of two allopolyploid individuals. Along the same logic, `Lynch.distance` will only be appropriate if the two homeologous genomes have no alleles in common at a given locus. If either of these distance measures are appropriate for your data, see the description of the `meandistance.matrix` function in sections 4.2.1 and 6.3.2. The `meandistance.matrix2` function is never appropriate under allopolyploid inheritance, since it assumes random segregation of alleles when calculating genotype probabilities. `Bruvo2.distance` is unlikely to be appropriate for an allopolyploid system, although I would encourage reading the paper[1] and thinking about it for yourself.

Assuming a distance matrix can be calculated using `meandistance.matrix`, all downstream analyses (principal coordinate analysis, clone assignment, genotype diversity) are appropriate.

The `estimatePloidy`, `assignClones`, `genotypeDiversity`, and `alleleDiversity` functions work equally well on autopolyploids and allopolyploids.

Both `simpleFreq` and `deSilvaFreq` work under the assumption of polysomic inheritance and should therefore not be used on allopolyploid data.

8 Treating microsatellite alleles as dominant markers

Both autopolyploid and allopolyploid microsatellite data can be converted to "allelic phenotypes" based on the presence and absence of alleles. Although much information is lost using this method, it can enable the user to perform a wider range of analyses, such as parentage analysis or AMOVA.

The `Lynch.distance` measure, described earlier, essentially treats alleles in this way. Alleles are assumed to be present in only one copy, and two alleles from two individuals are either identical or not. However, alleles are still grouped by locus and distances are averaged across all loci.

The `"genbinary"` class stores data in a binary presence/absence format, the same way that dominant data is typically coded. (See earlier description of the `genambig.to.genbinary` function in section 6.2.) This is intended to facilitate further analysis in R or other software that takes such a format. By default, `1` indicates that an allele is present, `0` indicates that an allele is absent, and `-9` indicates that the data point is missing. There are replacement functions to change these symbols, for example (continuing from section 5.3):

```
> Present(simgenB) <- "P"
> Absent(simgenB) <- 2
> Missing(simgenB) <- 0
> Genotypes(simgenB)[1:10, 1:6]

    loc1.100 loc1.102 loc1.104 loc1.106 loc1.108 loc1.110
A1  "2"      "2"      "2"      "P"      "2"      "P"
A2  "2"      "2"      "2"      "P"      "2"      "P"
A3  "P"      "P"      "2"      "P"      "2"      "2"
A4  "P"      "P"      "2"      "P"      "2"      "P"
```

```
A5  "2"      "2"      "2"      "P"      "2"      "2"
A6  "P"      "2"      "2"      "P"      "2"      "P"
A7  "2"      "2"      "2"      "2"      "P"      "2"
A8  "2"      "P"      "2"      "P"      "2"      "2"
A9  "2"      "2"      "2"      "2"      "2"      "2"
A10 "2"      "P"      "2"      "P"      "2"      "P"
```

If you want to further manipulate the format of the genotype matrix, you can assign it to a new object name and then make the desired edits.

```
> genmat <- Genotypes(simgenB)
> dimnames(genmat)[[2]] <- paste("M", 1:dim(genmat)[2], sep="")
> genmat[1:10, 1:10]
```

```
    M1  M2  M3  M4  M5  M6  M7  M8  M9  M10
A1  "2" "2" "2" "P" "2" "P" "P" "2" "2" "2"
A2  "2" "2" "2" "P" "2" "P" "2" "P" "2" "P"
A3  "P" "P" "2" "P" "2" "2" "2" "P" "2" "2"
A4  "P" "P" "2" "P" "2" "P" "2" "2" "2" "2"
A5  "2" "2" "2" "P" "2" "2" "P" "2" "2" "2"
A6  "P" "2" "2" "P" "2" "P" "2" "2" "2" "2"
A7  "2" "2" "2" "2" "P" "2" "P" "2" "2" "2"
A8  "2" "P" "2" "P" "2" "2" "2" "2" "2" "2"
A9  "2" "2" "2" "2" "2" "2" "P" "2" "2" "2"
A10 "2" "P" "2" "P" "2" "P" "P" "2" "2" "2"
```

As demonstrated previously, the `write.table` function can write the matrix to a text file for use in other software. The arguments for `write.table` allow the user to control which character is used to delimit fields, whether row and column names should be written to the file, and whether quotation marks should be used for character strings.

9 How to cite polysat

Clark, LV and Jasieniuk, M. POLYSAT: an R package for polyploid microsatellite analysis. *Molecular Ecology Resources* 11(3):562–566.

Feel free to email me at lvclark@illinois.edu with any questions, comments, or bug reports!

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] CHAMBERS, J. M. 2008. *Software for Data Analysis: Programming with R* Springer.

[3] DE SILVA, H. N, HALL, A. J., RIKKERINK, E., MCNEILAGE, M. A., and FASER, L. G. 2005. Estimation of allele frequencies in polyploids under certain patterns of inheritance. *Heredity*, 95, 327-334.

[4] 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.

[5] 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.

[6] GULDBRANDTSEN, B., TOMIUK, J. AND LOESCHCKE, B. 2000. POPDIST version 1.1.1: A program to calculate population genetic distance and identity measures. *Journal of Heredity*, 91, 178-179.

[7] 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.

[8] 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.

[9] JOMBART, T. 2008. adegenet: a R package for the multivariate analysis of genetic markers. *Bioinformatics*, 24, 1403-1405.

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

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

[16] SHANNON, C. E. 1948. A mathematical theory of communication. *Bell System Technical Journal*, 27, 379-423 and 623-656.

[17] SIMPSON, E. H. 1949. Measurement of diversity. *Nature*, 163, 688.

[18] TOMIUK, J. GULDGRANDTSEN, B. AND LOESCHCKE, B. 2009. Genetic similarity of polyploids: a new version of the computer program POPDIST (version 1.2.0) considers intraspecific genetic differentiation. *Molecular Ecology Resources*, 9, 1364-1368.

[19] TOONEN, R. J. and HUGHES, S. 2001. Increased Throughput for Fragment Analysis on ABI Prism 377 Automated Sequencer Using a Membrane Comb and STRand Software. *Biotechniques*, 31, 1320-1324.

[20] 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.