POLYSAT version 1.7 Tutorial Manual

Lindsay V. Clark <lvclark@illinois.edu>
University of Illinois, Urbana-Champaign
Department of Crop Sciences
https://github.com/lvclark/polysat/wiki
March 4, 2019

Contents

1 Introduction .................................................. 2
2 Obtaining and installing POLYSAT .................. 3
3 Workflow overview ........................................ 3
4 Getting Started: A Tutorial .......................... 6
  4.1 Creating a dataset ........................................ 6
  4.2 Data analysis and export .............................. 11
   4.2.1 Genetic distances between individuals .......... 11
   4.2.2 Working with subsets of the data ................ 13
   4.2.3 Population statistics ............................... 15
   4.2.4 Genotype data export ............................. 21
5 How data are stored in POLYSAT .................... 21
  5.1 The “genambig” class ................................ 21
  5.2 How ploidy data is stored: “ploidysuper” and subclasses .... 28
  5.3 The “gendata” and “genbinary” classes .............. 31
6 Functions for autopolyploid data ................. 34
  6.1 Data import ........................................... 34
  6.2 Data export ........................................... 37
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, population differentiation statistics such as $F_{ST}$, and polymorphic information content. 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 “polysattutorial.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/](http://www.r-project.org/). Once R is installed, POLYSAT can be installed and loaded by typing the following commands into the R console:

```r
> 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 inputting 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.
Import genotype data using a read function, or start from scratch using new and editGenotypes.

Run alleleDiversity, and/or convert to presence/absence (genbinary class) for analysis or export.

Check dataset with summary.

Add population and locus information (PopNames, PopInfo, Usatnts).

Analysis with assignClones or genotypeDiversity

Calculate genetic distances between individuals with mandistance.matrix.

Downstream analysis such as Principal Coordinate Analysis or Neighbor Joining.

Were ploidies imported with the dataset?

Do you know the ploidies?

Is organism allopolyploid?

Use alleleCorrelations, testAlGroups, and recodeAllopoly functions to recode the dataset as diploid or polysomic.

Export tetraploid data to ATetra, Tetra, or Tetrasat software.

Use estimatePloidy function.

reformatPloidies if necessary, then add ploidy info with Ploidies function.

yes

no

yes

no

Flowchart 1. Functions for allopolyploid or autopolyplodid data.
Continued from Flowchart 1.

Does each population have one, even numbered ploidy, and is the self fertilization rate known?

- Export to Structure, SPAGeDi, GenoDive, or POPDIST software.
- Calculate Polymorphic Information Content (PIC)
- Export allele frequencies to adegenet.
- Get distances between populations with calcPopDiff.
- Calculate genetic distances between individuals using meandistance.matrix2.
- Downstream analysis.
- Use simpleFreq to estimate allele frequencies.
- Use deSilvaFreq to estimate allele frequencies.
- Export allele frequencies to SPAGeDi.
- Analysis with assignClones or genotypeDiversity

Flowchart 2. Functions for polysomic or diploid 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] "C:/Users/lvclark/AppData/Local/Temp/Rtmpe40I0t/Rbuild17b4736a7bc4/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)
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.

```r
> 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):

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

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

```r
> 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:

```r
> 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 25 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.

```r
> 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 and you want to use the Bruvo distance, I recommend using meandistance.matrix2 rather than meandistance.matrix. meandistance.matrix2 will take longer to process, but will be more accurate because it models allele copy number. Additionally, 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.

```r
> 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.

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

[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"
```
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.

```r
> 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.

```r
> 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 disecting 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 $F_{ST}$ [15], Nei’s $G_{ST}$ [15, 16], Jost’s $D$ [9], and $R_{ST}$ [20] 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

<table>
<thead>
<tr>
<th>Genomes</th>
<th>loc1.100</th>
<th>loc1.102</th>
<th>loc1.104</th>
<th>loc1.106</th>
<th>loc1.108</th>
<th>loc1.110</th>
<th>loc1.112</th>
<th>loc1.114</th>
<th>loc1.116</th>
<th>loc1.118</th>
<th>loc1.null</th>
<th>loc2.143</th>
<th>loc2.146</th>
<th>loc2.149</th>
<th>loc2.152</th>
<th>loc2.155</th>
<th>loc2.158</th>
<th>loc2.161</th>
<th>loc2.164</th>
</tr>
</thead>
<tbody>
<tr>
<td>PopA</td>
<td>212</td>
<td>0.120299</td>
<td>0.120410</td>
<td>0.000000</td>
<td>0.219636</td>
<td>0.035917</td>
<td>0.142878</td>
<td>0.154229</td>
<td>0.125102</td>
<td>0.071184</td>
<td>0.010345</td>
<td>0.000000</td>
<td>0.162921</td>
<td>0.301323</td>
<td>0.024409</td>
<td>0.391124</td>
<td>0.058461</td>
<td>0.196465</td>
<td>0.028626</td>
</tr>
<tr>
<td>PopB</td>
<td>208</td>
<td>0.000000</td>
<td>0.169642</td>
<td>0.091277</td>
<td>0.066652</td>
<td>0.000000</td>
<td>0.128650</td>
<td>0.000000</td>
<td>0.125179</td>
<td>0.074908</td>
<td>0.034364</td>
<td>0.014014</td>
<td>0.091996</td>
<td>0.122846</td>
<td>0.110034</td>
<td>0.100000</td>
<td>0.584641</td>
<td>0.196465</td>
<td>0.028626</td>
</tr>
</tbody>
</table>
### simFst <- calcPopDiff(simfreq, metric = "Fst")
### simFst

<table>
<thead>
<tr>
<th></th>
<th>PopA</th>
<th>PopB</th>
<th>PopC</th>
</tr>
</thead>
<tbody>
<tr>
<td>PopA</td>
<td>0.0000000</td>
<td>0.05068795</td>
<td>0.05453103</td>
</tr>
<tr>
<td>PopB</td>
<td>0.05068795</td>
<td>0.0000000</td>
<td>0.07098261</td>
</tr>
<tr>
<td>PopC</td>
<td>0.05453103</td>
<td>0.07098261</td>
<td>0.0000000</td>
</tr>
</tbody>
</table>

### simFst12 <- calcPopDiff(simfreq, metric = "Fst", loci=c("loc1", "loc2"))
### simFst12

<table>
<thead>
<tr>
<th></th>
<th>PopA</th>
<th>PopB</th>
<th>PopC</th>
</tr>
</thead>
<tbody>
<tr>
<td>PopA</td>
<td>0.0000000</td>
<td>0.06004514</td>
<td>0.05597902</td>
</tr>
<tr>
<td>PopB</td>
<td>0.06004514</td>
<td>0.0000000</td>
<td>0.07356898</td>
</tr>
<tr>
<td>PopC</td>
<td>0.05597902</td>
<td>0.07356898</td>
<td>0.0000000</td>
</tr>
</tbody>
</table>

### simGst <- calcPopDiff(simfreq, metric = "Gst")
### simGst

<table>
<thead>
<tr>
<th></th>
<th>PopA</th>
<th>PopB</th>
<th>PopC</th>
</tr>
</thead>
<tbody>
<tr>
<td>PopA</td>
<td>-0.002364066</td>
<td>0.048319761</td>
<td>0.052403480</td>
</tr>
<tr>
<td>PopB</td>
<td>0.048319761</td>
<td>-0.002409639</td>
<td>0.068781401</td>
</tr>
<tr>
<td>PopC</td>
<td>0.052403480</td>
<td>0.068781401</td>
<td>-0.002785515</td>
</tr>
</tbody>
</table>
> simGst12 <- calcPopDiff(simfreq, metric = "Gst", loci=c("loc1", "loc2"))
> simGst12

<table>
<thead>
<tr>
<th></th>
<th>PopA</th>
<th>PopB</th>
<th>PopC</th>
</tr>
</thead>
<tbody>
<tr>
<td>PopA</td>
<td>-0.002364066</td>
<td>0.058073428</td>
<td>0.053930731</td>
</tr>
<tr>
<td>PopB</td>
<td>0.058073428</td>
<td>-0.002409639</td>
<td>0.071512283</td>
</tr>
<tr>
<td>PopC</td>
<td>0.053930731</td>
<td>0.071512283</td>
<td>-0.002785515</td>
</tr>
</tbody>
</table>

> simD <- calcPopDiff(simfreq, metric = "Jost's D")
> simD

<table>
<thead>
<tr>
<th></th>
<th>PopA</th>
<th>PopB</th>
<th>PopC</th>
</tr>
</thead>
<tbody>
<tr>
<td>PopA</td>
<td>-0.02382583</td>
<td>0.42191879</td>
<td>0.56407144</td>
</tr>
<tr>
<td>PopB</td>
<td>0.42191879</td>
<td>-0.01830667</td>
<td>0.63812043</td>
</tr>
<tr>
<td>PopC</td>
<td>0.56407144</td>
<td>0.63812043</td>
<td>-0.03000128</td>
</tr>
</tbody>
</table>

> simD12 <- calcPopDiff(simfreq, metric = "Jost's D", loci=c("loc1", "loc2"))
> simD12

<table>
<thead>
<tr>
<th></th>
<th>PopA</th>
<th>PopB</th>
<th>PopC</th>
</tr>
</thead>
<tbody>
<tr>
<td>PopA</td>
<td>-0.02429224</td>
<td>0.51180560</td>
<td>0.5604966</td>
</tr>
<tr>
<td>PopB</td>
<td>0.51180560</td>
<td>-0.01907365</td>
<td>0.6487600</td>
</tr>
<tr>
<td>PopC</td>
<td>0.5604966</td>
<td>0.64876003</td>
<td>-0.0274657</td>
</tr>
</tbody>
</table>

> simRst <- calcPopDiff(simfreq, metric = "Rst", object = simgen)
> simRst

<table>
<thead>
<tr>
<th></th>
<th>PopA</th>
<th>PopB</th>
<th>PopC</th>
</tr>
</thead>
<tbody>
<tr>
<td>PopA</td>
<td>4.239168e-17</td>
<td>9.085165e-02</td>
<td>3.612000e-02</td>
</tr>
<tr>
<td>PopB</td>
<td>9.085165e-02</td>
<td>-1.206474e-16</td>
<td>4.063042e-02</td>
</tr>
<tr>
<td>PopC</td>
<td>3.612000e-02</td>
<td>4.063042e-02</td>
<td>1.403716e-16</td>
</tr>
</tbody>
</table>

> simRst12 <- calcPopDiff(simfreq, metric = "Rst", object = simgen, loci=c("loc1", "loc2"))
> simRst12

<table>
<thead>
<tr>
<th></th>
<th>PopA</th>
<th>PopB</th>
<th>PopC</th>
</tr>
</thead>
<tbody>
<tr>
<td>PopA</td>
<td>6.358752e-17</td>
<td>1.370264e-01</td>
<td>3.723488e-02</td>
</tr>
<tr>
<td>PopB</td>
<td>1.370264e-01</td>
<td>-1.809712e-16</td>
<td>4.575589e-02</td>
</tr>
<tr>
<td>PopC</td>
<td>3.723488e-02</td>
<td>4.575589e-02</td>
<td>1.025220e-16</td>
</tr>
</tbody>
</table>
We can also calculate polymorphic information content (PIC) of each locus in order to gauge which loci will be most informative for future studies (higher numbers = more informative).

```
> PIC(simfreq)

   loc1       loc2       loc3
PopA 0.8398759 0.7709120 0.8033700
PopB 0.8057640 0.7127163 0.7445749
PopC 0.8245357 0.7810601 0.8424778
Overall 0.8886356 0.8616563 0.8623154
```

We can get a global estimate, rather than a pairwise estimate, of any population differentiation statistic, for example for $G_{ST}$:

```
> calcPopDiff(simfreq, metric = "Gst", global = TRUE)

[1] 0.07472934
```

For either pairwise or global population differentiation statistics, we can get bootstrapped estimates in order to determine a 95% confidence interval:

```
> gbootstrap <- calcPopDiff(simfreq, metric = "Gst", global = TRUE, + bootstrap = TRUE)
> quantile(gbootstrap, c(0.025, 0.975))

  2.5%       97.5%
0.06046229 0.10061991
```

```
> gbootstrap_pairwise <- calcPopDiff(simfreq, metric = "Gst", + bootstrap = TRUE)
> pairwise_CIs <- lapply(gbootstrap_pairwise, + function(x) quantile(x, c(0.025, 0.975)))
> names(pairwise_CIs) <- paste(rep(PopNames(simgen), + each = length(PopNames(simgen))), + rep(PopNames(simgen), + times = length(PopNames(simgen))), + sep = ".")
> pairwise_CIs
```
<table>
<thead>
<tr>
<th>Comparison</th>
<th>2.5%</th>
<th>97.5%</th>
<th>2.5%</th>
<th>97.5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{PopA.PopA}$</td>
<td>-0.002364066</td>
<td>-0.002364066</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{PopA.PopB}$</td>
<td>0.02881243</td>
<td>0.07977158</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{PopA.PopC}$</td>
<td>0.04584374</td>
<td>0.06201772</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{PopB.PopA}$</td>
<td>0.02881243</td>
<td>0.07977158</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{PopB.PopB}$</td>
<td>-0.002409639</td>
<td>-0.002409639</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{PopB.PopC}$</td>
<td>0.05408831</td>
<td>0.08893625</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{PopC.PopA}$</td>
<td>0.04584374</td>
<td>0.06201772</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{PopC.PopB}$</td>
<td>0.05408831</td>
<td>0.08893625</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{PopC.PopC}$</td>
<td>-0.002785515</td>
<td>-0.002785515</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 diploid 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.

```r
> 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.

```r
> 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.

```r
> mydataset
```

---

22
Insert dataset description here.
Missing values: -9

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

SSR motif lengths:
  none

Ploidies:
  none

PopNames:
  none

PopInfo:
  none

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)
```r
[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)


> 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

25
> *Missing*(mydataset)

[1] -9

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

<table>
<thead>
<tr>
<th></th>
<th>L1</th>
<th>L2</th>
<th>L3</th>
</tr>
</thead>
<tbody>
<tr>
<td>indA</td>
<td>Numeric,3</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>indB</td>
<td>Numeric,2</td>
<td>-1</td>
<td>Numeric,3</td>
</tr>
<tr>
<td>indC1</td>
<td>Numeric,4</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>indD</td>
<td>Numeric,3</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>indE</td>
<td>Numeric,3</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>indF</td>
<td>Numeric,2</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>L1</th>
<th>L2</th>
<th>L3</th>
</tr>
</thead>
<tbody>
<tr>
<td>indA</td>
<td>Numeric,3</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>indB</td>
<td>Numeric,2</td>
<td>-1</td>
<td>Numeric,3</td>
</tr>
<tr>
<td>indC1</td>
<td>Numeric,4</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>indD</td>
<td>Numeric,3</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>indE</td>
<td>Numeric,3</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>indF</td>
<td>Numeric,2</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

> 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.
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.

```r
> moredata <- new("genambig", samples=c("indG", "indH"), loci=Loci(mydataset))
> Usatnts(moredata) <- Usatnts(mydataset)
> Description(moredata) <- "Kings Canyon"
> PopNames(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
```

**Tutorial, part 2.**

Missing values: -1

**Genotypes:**

```
   L1   L2   L3
indA 122/124/128 -1 -1
```
indB 124/126  -1 150/154/160
indC1 120/126/128/130  -1 -1
indD 122/124/130  -1 -1
indE 128/130/132  -1 -1
indF 126/130  -1 -1
indG 126/130/136/138  -1 -1
indH 124/126/128  -1 -1

SSR motif lengths:
L1 L2 L3
2 2 2

Ploidies:
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

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

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

### 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 "ploidy-super" 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.

<table>
<thead>
<tr>
<th>Class</th>
<th>Format</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>ploidyone</td>
<td>unnamed vector of length one</td>
<td>uniform ploidy for entire dataset</td>
</tr>
<tr>
<td>ploidysample</td>
<td>vector indexed by sample</td>
<td>samples vary in ploidy</td>
</tr>
<tr>
<td>ploidylocus</td>
<td>vector indexed by locus</td>
<td>loci vary in copy number</td>
</tr>
<tr>
<td>ploidymatrix</td>
<td>matrix indexed by sample then locus</td>
<td>different samples have different numbers of copies of different loci</td>
</tr>
</tbody>
</table>

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 reformat-Ploidies 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 \texttt{?reformatPloidies} for more information on how to change formats when there is already data in the \texttt{Ploidies} slot.

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

\begin{verbatim}
> Ploidies(ploidyexample)["ind1", "loc1"]
[1] 4
\end{verbatim}

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

\begin{verbatim}
> Ploidies(ploidyexample, "ind1", "loc1")
  loc1
  ind1 4

> ploidyexample <- reformatPloidies(ploidyexample, output="one")
> Ploidies(ploidyexample, "ind1", "loc1")
[1] 4
\end{verbatim}

5.3 The “gendata” and “genbinary” classes

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

A second subclass of "gendata" is "genbinary". This class also has a \texttt{Genotypes} slot, but formatted as a matrix indicating the presence and absence of alleles. (See \texttt{?genbinary-class} for more details.) It also adds a slot called \texttt{Present} and one called \texttt{Absent} to indicate the symbols used to represent the presence or absence of the alleles, the same way the \texttt{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.
```r
> simgenB <- genambig.to.genbinary(simgen)
> Genotypes(simgenB, samples=paste("A", 1:20, sep=""), loci="loc1")

<table>
<thead>
<tr>
<th></th>
<th>loc1.100</th>
<th>loc1.102</th>
<th>loc1.104</th>
<th>loc1.106</th>
<th>loc1.108</th>
<th>loc1.110</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A6</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>A8</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A10</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A11</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A14</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A15</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A16</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A17</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A18</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A19</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A20</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>loc1.112</th>
<th>loc1.114</th>
<th>loc1.116</th>
<th>loc1.118</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A7</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A10</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A11</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

32
> PopInfo(simgenB)[Samples(simgenB, ploidies=2)]

<table>
<thead>
<tr>
<th></th>
<th>A5</th>
<th>A7</th>
<th>A8</th>
<th>A9</th>
<th>A12</th>
<th>A13</th>
<th>A17</th>
<th>A18</th>
<th>A21</th>
<th>A22</th>
<th>A23</th>
<th>A25</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>A27</td>
<td>A30</td>
<td>A31</td>
<td>A32</td>
<td>A35</td>
<td>A40</td>
<td>A44</td>
<td>A45</td>
<td>A47</td>
<td>A50</td>
<td>A52</td>
<td>A53</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A54</td>
<td>A55</td>
<td>A56</td>
<td>A58</td>
<td>A59</td>
<td>A65</td>
<td>A67</td>
<td>A71</td>
<td>A72</td>
<td>A73</td>
<td>A74</td>
<td>A75</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A77</td>
<td>A78</td>
<td>A80</td>
<td>A84</td>
<td>A87</td>
<td>A88</td>
<td>A91</td>
<td>A93</td>
<td>A95</td>
<td>A96</td>
<td>A100</td>
<td>B1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>B4</td>
<td>B7</td>
<td>B8</td>
<td>B9</td>
<td>B13</td>
<td>B14</td>
<td>B15</td>
<td>B16</td>
<td>B17</td>
<td>B20</td>
<td>B25</td>
<td>B27</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>B30</td>
<td>B32</td>
<td>B34</td>
<td>B35</td>
<td>B36</td>
<td>B39</td>
<td>B41</td>
<td>B49</td>
<td>B50</td>
<td>B52</td>
<td>B54</td>
<td>B57</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>B58</td>
<td>B59</td>
<td>B61</td>
<td>B62</td>
<td>B64</td>
<td>B65</td>
<td>B68</td>
<td>B72</td>
<td>B73</td>
<td>B74</td>
<td>B77</td>
<td>B80</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>B82</td>
<td>B84</td>
<td>B85</td>
<td>B86</td>
<td>B89</td>
<td>B93</td>
<td>B94</td>
<td>B96</td>
<td>B97</td>
<td>B98</td>
<td>B99</td>
<td>C1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td>C4</td>
<td>C6</td>
<td>C7</td>
<td>C8</td>
<td>C10</td>
<td>C11</td>
<td>C14</td>
<td>C16</td>
<td>C17</td>
<td>C20</td>
<td>C21</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>C23</td>
<td>C25</td>
<td>C27</td>
<td>C28</td>
<td>C31</td>
<td>C32</td>
<td>C36</td>
<td>C37</td>
<td>C38</td>
<td>C39</td>
<td>C40</td>
<td>C44</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>C46</td>
<td>C47</td>
<td>C48</td>
<td>C50</td>
<td>C56</td>
<td>C57</td>
<td>C59</td>
<td>C61</td>
<td>C64</td>
<td>C67</td>
<td>C68</td>
<td>C71</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>C74</td>
<td>C75</td>
<td>C76</td>
<td>C77</td>
<td>C79</td>
<td>C80</td>
<td>C82</td>
<td>C83</td>
<td>C84</td>
<td>C85</td>
<td>C86</td>
<td>C87</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>C90</td>
<td>C92</td>
<td>C93</td>
<td>C95</td>
<td>C96</td>
<td>C98</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

33
The "genbinary" class exists to facilitate the import and export of genotype data formatted in a binary presence/absence format, for example:

```r
> 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 there 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 (excluding ancient autopolyploids that have undergone diploidization), 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. If you have data from an allopolyploid or diploidized autopolyploid organism, you may also want to see the vignette “Assigning alleles to isoloci in POLYSAT”.

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, 17, 8], SPAGeDi [7], GenoDive
and POPDIST [6][21].

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:

```r
> 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.

```r
> ?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 “GeneMapperExample.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:

```r
> 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 [22]. Since this format uses one row per individual, the modified format for POLYSAT includes a column to contain population information.

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

<table>
<thead>
<tr>
<th>Pop</th>
<th>Ind</th>
<th>GSSR28</th>
<th>GSSR76</th>
<th>GSSR90</th>
</tr>
</thead>
<tbody>
<tr>
<td>NC1</td>
<td>a</td>
<td>134/140/154*</td>
<td>135/144/158*</td>
<td>201/211</td>
</tr>
<tr>
<td>NC1</td>
<td>b</td>
<td>140/154/172*</td>
<td>106/135/144/158*</td>
<td>205/211/213*</td>
</tr>
<tr>
<td>NC1</td>
<td>c</td>
<td>140/154</td>
<td>107/136/145/158*</td>
<td>199/207</td>
</tr>
<tr>
<td>VA1</td>
<td>a</td>
<td>0</td>
<td>145/149</td>
<td>199/207/211*</td>
</tr>
<tr>
<td>VA1</td>
<td>b</td>
<td>154</td>
<td>106/145</td>
<td>201/207</td>
</tr>
<tr>
<td>VA1</td>
<td>c</td>
<td>138/172</td>
<td>107/136/149*</td>
<td>207/211/213*</td>
</tr>
<tr>
<td>VA1</td>
<td>d</td>
<td>138/140/154*</td>
<td>135/158</td>
<td>207/213</td>
</tr>
</tbody>
</table>
```

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

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

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

<table>
<thead>
<tr>
<th></th>
<th>ABC1.123</th>
<th>ABC1.126</th>
<th>ABC1.129</th>
<th>ABC1.132</th>
<th>ABC1.135</th>
<th>ABC2.201</th>
</tr>
</thead>
<tbody>
<tr>
<td>ind1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ind2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ind3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>ABC2.203</th>
<th>ABC2.205</th>
<th>ABC2.207</th>
<th>ABC2.209</th>
</tr>
</thead>
<tbody>
<tr>
<td>ind1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ind2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ind3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

> 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.

```r
> 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 = ",",)

39
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:

```r
> 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 [12]. (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.

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

```r
> 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.

```r
> 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 [14], 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

<table>
<thead>
<tr>
<th></th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>A5</th>
<th>A6</th>
<th>A7</th>
<th>A8</th>
<th>A9</th>
<th>A10</th>
<th>A11</th>
<th>A12</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>A13</td>
<td>A14</td>
<td>A15</td>
<td>A16</td>
<td>A17</td>
<td>A18</td>
<td>A19</td>
<td>A20</td>
<td>A21</td>
<td>A22</td>
<td>A23</td>
<td>A24</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>A25</td>
<td>A26</td>
<td>A27</td>
<td>A28</td>
<td>A29</td>
<td>A30</td>
<td>A31</td>
<td>A32</td>
<td>A33</td>
<td>A34</td>
<td>A35</td>
<td>A36</td>
</tr>
</tbody>
</table>
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.

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

<table>
<thead>
<tr>
<th></th>
<th>loc1</th>
<th>loc2</th>
<th>loc3</th>
</tr>
</thead>
<tbody>
<tr>
<td>PopA</td>
<td>8</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>PopB</td>
<td>7</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>PopC</td>
<td>9</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>overall</td>
<td>10</td>
<td>8</td>
<td>11</td>
</tr>
</tbody>
</table>
```

```r
> 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 calcPopDiff 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 [15]. 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:

```r
> 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 <- calcPopDiff(simfreqSimple, metric = "Fst")
> simFstSimple
```

46
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[10] 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 may want to calculate statistics based on the frequencies of genotypes in your populations. Two popular statistics for this, the Shannon index [18] and Simpson index [19], 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 individually 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

<table>
<thead>
<tr>
<th>loc1</th>
<th>loc2</th>
<th>loc3</th>
<th>overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>PopA</td>
<td>3.304798</td>
<td>2.654090</td>
<td>3.379927</td>
</tr>
<tr>
<td>PopB</td>
<td>3.269491</td>
<td>2.582981</td>
<td>2.913112</td>
</tr>
<tr>
<td>PopC</td>
<td>3.373131</td>
<td>3.009567</td>
<td>3.415639</td>
</tr>
</tbody>
</table>

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

> simdiv2

<table>
<thead>
<tr>
<th>loc1</th>
<th>loc2</th>
<th>loc3</th>
<th>overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>PopA</td>
<td>0.037374</td>
<td>0.083030</td>
<td>0.033601</td>
</tr>
<tr>
<td>PopB</td>
<td>0.037029</td>
<td>0.088848</td>
<td>0.056970</td>
</tr>
<tr>
<td>PopC</td>
<td>0.032727</td>
<td>0.053737</td>
<td>0.031276</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>loc1</th>
<th>loc2</th>
<th>loc3</th>
<th>overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>PopA</td>
<td>0.024383</td>
<td>0.059151</td>
<td>0.021139</td>
</tr>
<tr>
<td>PopB</td>
<td>0.022950</td>
<td>0.061151</td>
<td>0.042167</td>
</tr>
<tr>
<td>PopC</td>
<td>0.021352</td>
<td>0.034039</td>
<td>0.020075</td>
</tr>
</tbody>
</table>

> simdiv2 + 2*simdiv2var^0.5

<table>
<thead>
<tr>
<th>loc1</th>
<th>loc2</th>
<th>loc3</th>
<th>overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>PopA</td>
<td>0.050365</td>
<td>0.106909</td>
<td>0.046032</td>
</tr>
<tr>
<td>PopB</td>
<td>0.051108</td>
<td>0.116544</td>
<td>0.071715</td>
</tr>
<tr>
<td>PopC</td>
<td>0.044119</td>
<td>0.073436</td>
<td>0.042180</td>
</tr>
</tbody>
</table>

48
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 autopolyplloid, 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. See the separate vignette “Assigning alleles to isoloci in polysat” to learn more about how to determine which alleles belong to which genomes. The functions processDatasetAllo and recodeAllopoly can, respectively, assign alleles to isoloci and recode the dataset so that each marker is split into multiple isoloci according to allele assignments.

If you are not able split microsatellite markers into independently-segregating isoloci, the following functionality is available for allopolyploids in POLYSAT:

7.1 Data import and export

Data can be formatted for the software Tetrasat [13], Tetra [11], and ATetra [23] 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")
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.

```r
> write.ATetra(simgen, samples=Samples(simgen, ploidies=4), file="simgenAT.txt")
Missing data: B48 loc2
Missing data: A42 loc3
Missing data: C22 loc3
```

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 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.matrix** 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):

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

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

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


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

References


