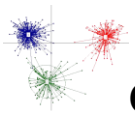


GPoppin (an R package) Documentation and Example

By Steven Micheletti, 6/29/2018

I. Introduction to the package

Purpose: GPoppin was created to efficiently organize and analyze genetic data that are in Genepop or Progeny export formats. It was designed with GT-sequencing data in mind, but can effectively handle large genome-wide datasets as well. One large benefit is its ability to convert Progeny database exports (1 allele per column format) into Genepop format and subsequently filter the Genepop file by many criteria. It also works well as precursor to software packages that require Genepop format.

Installation: GPoppin is intended to be installed on desktop computers, but also functions on servers. The package is available on https://github.com/StevenMicheletti/GPoppin.

1) Download GPoppin.tar
2) Open R (or Rstudio) and switch to the directory where Gpoppin.tar is downloaded

```
> setwd("C:/Users/person/Downloads")
```

3) Perform a local install:

```
> install.packages("GPoppin.tar", repos=NULL, type="source")
```

4) Load the package:

```
> library("GPoppin")
```

If there are any errors, make sure that the prerequisite packages are installed below:

Prerequisites:

The following packages should be installed on R prior to using GPoppin:

```
>install.packages(c('adegenet', 'data.table', 'ggplot2', 'hierfstat',
'pegas'))
```

Alternatively, if *devtools* is installed, GPoppin can be unpacked and installed:

```
> install("GPoppin",dependencies = TRUE)
```

OR installed from GitHub:

```
> install_github("StevenMicheletti/GPoppin/install")
```

Overview of functions:

prog2gp - Convert Progeny exports (1 allele per columns format) into Genepop format.
fixformat - Fix special characters and locus names in Genepop that analysis tools have issues with.
popgen - Perform a suite of analyses with adegenet, generate summaries, QC loci.
sortloci - Remove or keep loci based on external list.
Buildpops - Define populations and individuals that belong to them. Good for subsetting individuals.
matchloci - Rearrange locus order and content based on a user-provided list.

genosim - Simulate individual genotypes based on allele frequencies. Can incorporate Ho-He.

Function details:

Documentation is provided within GPoppin and can be accessed with ?functionname

prog2gp

prog2gp uses an algorithm to automatically determine where genotype information starts in a Progeny export and converts those genetic data into a Genepop format. Specifically, it converts A into 102, C into 104, G into 106, T into 108, indels into 100, X into 006, and Y into 009. Missing data are converted into 000.

prog2gp has many error checking and correction features such as fixing characters in locus names that will interfere with other popular R packages (*i.e.*, adegenet). It can also remove individuals that do not meet a genotype success threshold.

Description

Convert a progeny export csv, 1 allele per column format, to Genepop format.

Usage

```
prog2gp(infile, rem.indels = FALSE, rem.sex = FALSE, fix.loc = TRUE,
  rem.thres = 0, split.pop = FALSE, rem.dup = TRUE, examp = FALSE)
```

Arguments

`infile`	Input progeny export table. csv or txt. REQUIRED.
`rem.indels`	Remove indels from dataset. Default = FALSE.
`rem.sex`	Remove sex markers from dataset. Default = FALSE.
`fix.loc`	Fix problematic locus names that contain invalid characters for R packages. Default = TRUE.
`rem.thres`	Remove loci that are not genotyped in N percent of individuals. Default = 0.
`split.pop`	Attempt to split genepop file into pops based on individual names. Default = FALSE.
`rem.dup`	Remove duplicated individuals by name. Default = TRUE.
`examp`	Use example file. Default = FALSE.
`first.col`	Genotypes start at the second column, after individual names. Default = FALSE.

Examples

```
prog2gp(infile = "progeny_export.csv", rem.indels=FALSE, rem.sex=FALSE,
```

```
fix.loci=TRUE, rem.thres=0, split.pop=TRUE, rem.dup=TRUE, examp=FALSE, first.col =
FALSE)
```

fixformat

fixformat checks Genepop files for odd characters and whitespace issues and attempts to fix the format. It is intended to be used on Genepop files that are not generated by GPoppin.

Description

Fixes common whitespace and invalid character errors in genepop files

Usage

```
fixformat(gpfile)
```

Arguments

`gpfile`	Genepop file. REQUIRED.

Examples

```
fixformat(gpfile="Populations.gen")
```

popgen

popgen performs many analyses on a Genepop file with the intention of providing summary statistics (by population and by locus) and flagging loci that are showing signals of sequencing errors. Multiple figures and tables are generated based on arguments used.

Description

Perform multiple population genetic analyses on a Genepop file and flag loci that don't meet criteria

Usage

```
popgen(gpfile, maf = TRUE, geno.s = TRUE, fail.pops = TRUE,
  pop.thres = 1, loc.stats = TRUE, maf.fz = 0.05, fail.loci = TRUE,
  pop.table = TRUE, hwe.calc = FALSE, iter = 100, p.pop = 0.33,
  p.thres = 0.05, calc.fst = FALSE, pca = TRUE, fz.table = TRUE)
```

Arguments

`gpfile`	Genepop filename. REQUIRED.
`maf`	Create minor allele frequency table for each locus. Default=TRUE.
`geno.s`	Determine genotype success of each locus. Default=TRUE.
`pop.thres`	Number of populations a locus must be present in. If fail.pops = TRUE. Default = 1.
`loc.stats`	Calculate stats such as He, Ho, He-Ho, FIS for each locus. Default = TRUE.
`maf.fz`	Flag loci that have a minor allele frequency less than N. Default = 0.05.
`pop.table`	Create a summary table of population level analyses (Ho,He, Ho-He, FIS, GenoS,N). Default = TRUE.
`hwe.calc`	Calculate Hardy-Weinberg equilibrium for each locus. Default = FALSE.
`iter`	Number of iterations to perform in HWE analysis. If hwe.calc = TRUE. Default = 100.
`p.pop`	Flag loci that are out of HWE in N proportion of populations. If hwe.calc = TRUE. Default = 0.33.
`p.thres`	p-value threshold for HWE tests. If hwe.calc = TRUE. Default = 0.05.
`calc.fst`	Calculate FST matrix (by population). Default = FALSE.
`pca`	Perform an individual-level principal component analysis. Default = TRUE.
`fz.table`	Create an allele frequency table for each locus. Default = TRUE.
`fail.pops`	Flag outlier He-Ho by MAF loci (fin). Default = TRUE.

Examples

```
popgen(gpfile="Populations.gen", maf=TRUE, geno.s=TRUE, fail.pops=TRUE,
pop.thres=1, loc.stats=TRUE, maf.fz=0.05, fail.pops=TRUE, pop.table=TRUE,
hwe.calc=TRUE, iter=100, p.pop=0.33, p.thres=0.05, calc.fst=TRUE, pca=TRUE,
fz.table=TRUE)
```

sortloci

sortloci will add or remove loci from a Genepop file based on a user-provided list. It is a great function to subset loci based on *popgen*, or simply remove or retain markers important for specific questions.

Description

Keep or remove a list of loci from a Genepop file

Usage

```
sortloci(gpfile, locus.file, option)
```

Arguments

`gpfile`	Name of genepop file that will be filtered. REQUIRED.
`locus.file`	Name of file that has list of locus names. REQUIRED.
`option`	Indicate if you want to keep or remove loci in the locus.file. Either "keep" or "remove" .REQUIRED.

Examples

```
sortloci(gpfile = "Population.gen", locus.file= "significant_snps.txt", option =
"keep")
```

buildpops

buildpops assigns individuals to different populations based on a user-provided table. Individuals can quickly be arranged into different groups and individuals that are not provided in the table are removed. The table has a population in each row, with the population number (arbitrary) in the first column, and each individual belonging to that population in subsequent columns

Description

Sort, filter, and rearrange Genepop populations by individual names

Usage

```
buildpops(gpfile, pop.file)
```

Arguments

`gpfile`	Name of Genepop file. REQUIRED.
`pop.file`	Name of file with population designation. REQUIRED.

Examples

```
buildpops (gpfile="Populations.gen", pop.file="5_pops.txt")
```

matchloci

matchloci reorganizes Genepop files by a user-provided order of locus names. Its main intention is to reorganize a Genepop file to match a previously created Genepop file with shared loci so that the two files can be combined. Any mismatching loci will be filled with missing data (000)

Description

Reorganizes loci in Genepop file to match the order of another list of loci.

Usage

```
matchloci(gpfile, loci.file)
```

Arguments

`gpfile`	Genepop file name to be reorganized. REQUIRED.
`loci.file`	File name of list of loci in desired order. REQUIRED.

Examples

```
matchloci(gpfile="Population.gen", loci.file="Loci_by_significance.txt")
```

genosim

genosim simulates genotype data from allele frequency tables. It can also use Expected heterozygosity-Observed heterozygosity information to more accurately reflect deviations from HWE. Both of the input tables are generated, by default, by *popgen*.

Description

Simulate genepop genotypes using allele frequency tables and HW disequilibrium

Usage

```
genosim(infile, iter = 100, hwefile = NULL)
```

Arguments

`infile`	Input allele frequency table (from GPoppin::popgen). REQUIRED.
`iter`	Number of individuals to simulate per population. Default = 100.
`hwefile`	Input Hardy-Weinberg disequilibrium table (from GPoppin::popgen) Default = NULL.

Examples

```
genosim(infile="Populations_freqtable.txt ", iter = 200, hwefile =
"Populations_locs_dis.txt")
```

II. Detailed use of GPoppin

Synopsis

To illustrate the workflow of *GPoppin*, I will use an example dataset of steelhead trout from 3 collection localities sequenced with CRITFC's Omy379 GT-seq panel. This 379-marker panel consist mostly of sex markers, neutral population differentiating markers, and maturation-related markers.

Converting from Progeny

In many cases, users will already have a Genepop file that has been created by a package such as STACKS, PGDSPIDER, PLINK, *etc*. However, if a Progeny database export is used, both collection data and genotype data for each individual will be provided in a large table. *prog2gp* takes Progeny export data (1 allele per column format) and converts it into a Genepop format.

The Progeny export I use in the example is a large tab-delimited table: 665 rows by 857 columns. Due to the nature of the export, some rows are blank, and many of the columns are blank. The export also contains individuals that are not genotyped at all. The first 99 columns are collection data, and the remaining 758 columns are alleles for the 379 markers. *prog2gp* is designed to manage this complicated table and extract solely the genotype of each individual.

Thus, *prog2gp* needs a Progeny export which 3 requirements:

1) There must be an Individual name column
2) Alleles must be A,T,C,G,X,Y or - . A/B allele format isn't supported
3) Each marker must be split into 2 columns, 1 for each allele (diploid)

If these requirements are met, we can then run prog2gp:

```
> prog2gp(infile ="gp_example.csv", rem.indels=FALSE, rem.sex=FALSE,
fix.loci=TRUE, rem.thres=.20, split.pop=TRUE, rem.dup=TRUE, examp=FALSE)
```

And R outputs:

```
[1] "Loading file and determining locus start column..."
[1] "Starting locus is OmyY1_2SEXY.A1 at column 46 of gpops"
[1] "Starting loci number = 378"
[1] "378 loci retained after any filters"
[1] "366 individuals initially"
[1] "0 individual IDs are duplicates and have been resolved"
[1] "Duplicated individual: "
[1] "159 individuals removed due to low genotyping (Have 80% missing data or more)"
[1] "Problematic characters in locus names were replaced with $"
[1] "Any occurrence of pop in locus names were replaced with pep"
[1] "Writing file: gp_example.gen"
[1] "Finished without errors"
```

```
   user  system elapsed
   1.64    0.01    1.66
```

The script took 1.66 seconds to run. After removing blank columns, it determined that OmyY1_2SEXY.A1 was the start of the genotype information. If any errors are encountered, it is a good idea to ensure that the correct start column is being used. If errors still occur, the user can physically modify the export so that individual names are on column 1, and subsequent columns are genotype info and setting first.ocl=TRUE.

Next, *prog2gp* found that 378/379 markers had genotype information available (1 marker that only amplified in hybrids was empty). 366 individuals were included in the table, and no individual IDs were duplicated. If duplicates are found, the names of the individuals will be printed. Since I set rem.thres=0.20, *prog2gp* only retained individuals that had at least 80% genotyping success. Individuals with 20% missing data (or no genotype data at all) were removed, leading to 159 individuals. Problematic characters in locus names such as dashes (-) periods (.) and the word "pop" are replaced accordingly to avoid error in subsequent analysis packages.

Since there are not many loci, I can open "gp_example.gen" to take a look at the Genepop output. The heading reads:

A=102,C=104,G=106,T=108,-=100,X=006,Y=009; Created by prog2gp on 2018-06-18 11:19:49

Which serves as a key if I ever forget what the numbers represent in the future. This key also was designed to match the current format of GSI and PBT baselines.

Scrolling down, everything looks normal except there is only one population identified in the Genepop file. split.pop=TRUE attempts to split populations by naming convention, but often the naming convention doesn't allow this. For instance, my individuals all have the prefix OmyWSR16- so *prog2gp* thinks they all belong to the same population. If I want to utilize this feature in the future, I could rename individuals to follow the format: Pop1_01, Pop1_02, Pop2_01, Pop2_02, *etc.*

<u>Organizing individuals</u>

Since I know which individuals belong to which collection locality, I will utilize *buildpops* to organize individuals into different populations. To do this, I simply make a table of the population in each row followed by the individuals that belong to that population:

```
> pop_sort.txt
     1 OmyWSR16-0030 OmyWSR16-0015 OmyWSR16-0017
     2 OmyWSR16-0007 OmyWSR16-0009 OmyWSR16-0011 OmyWSR16-0003 OmyWSR16-0013
     3 OmyWSR16-0204 OmyWSR16-0349
```

This table can quickly be made in Excel or R. In the example above, I'm only utilizing 10 individuals; the rest will be removed from the Genepop file. The population number (1-3) is arbitrary and simply will determine how the populations are ordered in the Genepop file.

Now I simply run *buildpops* with the Genepop file and pop_sort.txt as inputs:

```
> buildpops (gpfile="gp_example.gen ", pop.file="pop_sort.txt")
```

R prints:

```
[1] "Reading  genepop file to get population and locus information"
[1] "Reading and splitting genepop file"
[1] "Making 3 Population(s)"
[1] "Compiling populations:"
[1] "OmyWSR16-0030" "OmyWSR16-0015" "OmyWSR16-0017"
[1] "Compiling populations:"
[1] "OmyWSR16-0007" "OmyWSR16-0009" "OmyWSR16-0011" "OmyWSR16-0003"
[5] "OmyWSR16-0013"
[1] "Compiling populations:"
[1] "OmyWSR16-0204" "OmyWSR16-0349"
[1] "Loading compiled list"
[1] "Done! look for pop_sort file"
```

The output file is now "gp_example_pop_sort.gen." GPoppin automatically tags on extra identifiers to the Genepop filename after each function is run. In some cases, it may be beneficial to rename the output files after they are produced.

Looking through gp_example_pop_sort.gen, we see that the 10 individuals from pop_sort.txt belong to the populations we specified.

Subsetting Loci

Now that I have populations identified, I want to perform population analyses. However, I do not want to use all of my loci – I want to subset by the population informative ones. Additionally, I am interested in maturation genotypes in these individuals and want to make a separate Genepop file of only these adaptive markers. All of this can easily be done with *sortloci*.

First, I create a list of population informative loci. In this case, 269 of the loci are informative for population differentiation:

> pop_loci.txt

```
Omy_RAD36$67
Omy_RAD619$59
Omy_RAD739$59
Omy_RAD1186$59
Omy_RAD1751$18
Omy_RAD256$78
Omy_RAD2976$26
Omy_RAD3209$10
Omy_RAD7016$31
Omy_RAD9408$71
Omy_RAD12439$64
Omy_RAD13073$16
Omy_RAD14269$30
Omy_RAD14541$72
…
```

The order of the loci does not matter, as long as the names match those in the Genepop file.
Now, I run *sortloci* to subset the file

```
> sortloci(gpfile = "gp_example_pop_sort.gen", locus.file= "pop_loci.txt",
  option = "keep")
```

If these were blacklisted loci I would switch the option to "remove." In this case, I'm going to keep the list of loci that I provide. The script runs for a second and produces "gp_example_pop_sort_loci_sort.gen." The name is getting out of hand, so I simply rename it to "gp_example_pop.gen"

Now, for the adaptive markers. I target the known maturation loci as a separate file:

>mat_keep.txt

Omy_GREB1_05
Omy_GREB1_06
Omy_GREB1_09
Omy_RAD47080$54
Omy_RAD52458$17

…and run *sortloci* using this file

```
> sortloci(gpfile = "gp_example_pop_sort.gen", locus.file= " mat_keep.txt ", option
= "keep")
```

… producing another "gp_example_pop_sort_loci_sort.gen" which I rename to gp_example_mat.gen

Generating Population Statistics

Our data are now sorted into two distinct marker sets set up to address two different questions. We can analyze them with *popgen*. *popgen* requires that >1 population is included in the Genepop file. If, for some reason, there is only 1 population in a Genepop file, and the user still wants some of the stats generated, they can simply use a dummy population or duplicate the single population.

popgen does a lot, from flagging loci, to generating population stats, to generating locus stats, to performing some structure-related analyses, to generating other helpful plots. Let's run the population differentiation loci first.

```
> popgen(gpfile="gp_example_pop.gen", maf=TRUE, geno.s=TRUE, fail.pops=TRUE,
pop.thres=1,
loc.stats=TRUE, maf.fz=0.05, pop.table=TRUE, hwe.calc=TRUE,
iter=100, p.pop=0.33, p.thres=0.05, calc.fst=TRUE, pca=TRUE, fz.table=TRUE)
```

In this example, I am having *popgen* perform all possible analyses. By default, it turns the more computationally intensive analyses off (*e.g.,* HWE and FST). Since I have a small dataset, it will run quickly.

popgen produces some status updates as it runs. With all options on, it also produces ~25 files corresponding to all the analyses:

PCA files:

popgen performs a PCA on both populations and individuals. It also produces the loading scores (squared) for the first axis of the PCA and orders it by the loci with the highest scores. This is helpful when trying to determine loci that, say, lead to the most differentiation between populations.

Tables:

_maft: Minor allele frequency for each locus.
_freqtable : Allele frequencies. This file is also used by *genosim.*
_pop_table: Population genetic analyses (Ho, He, FIS, Genotyping success) by population.
_ho/he/FIS: Stats by locus.
_dis: Hardy-Weinberg disequilibrium (Ho-He) by locus. This file can be used by genosim
_genos: Genotyping success by locus.
_fin: Fin plot. Looks at Ho-He by Minor Allele Frequency .
.fst : Population FST matrix.
_hwe: HWE p-value table by locus.
_HWEcomp: Comparison of correction methods for HWE. Shows the proportion of populations that fail.

Files Flagging loci:

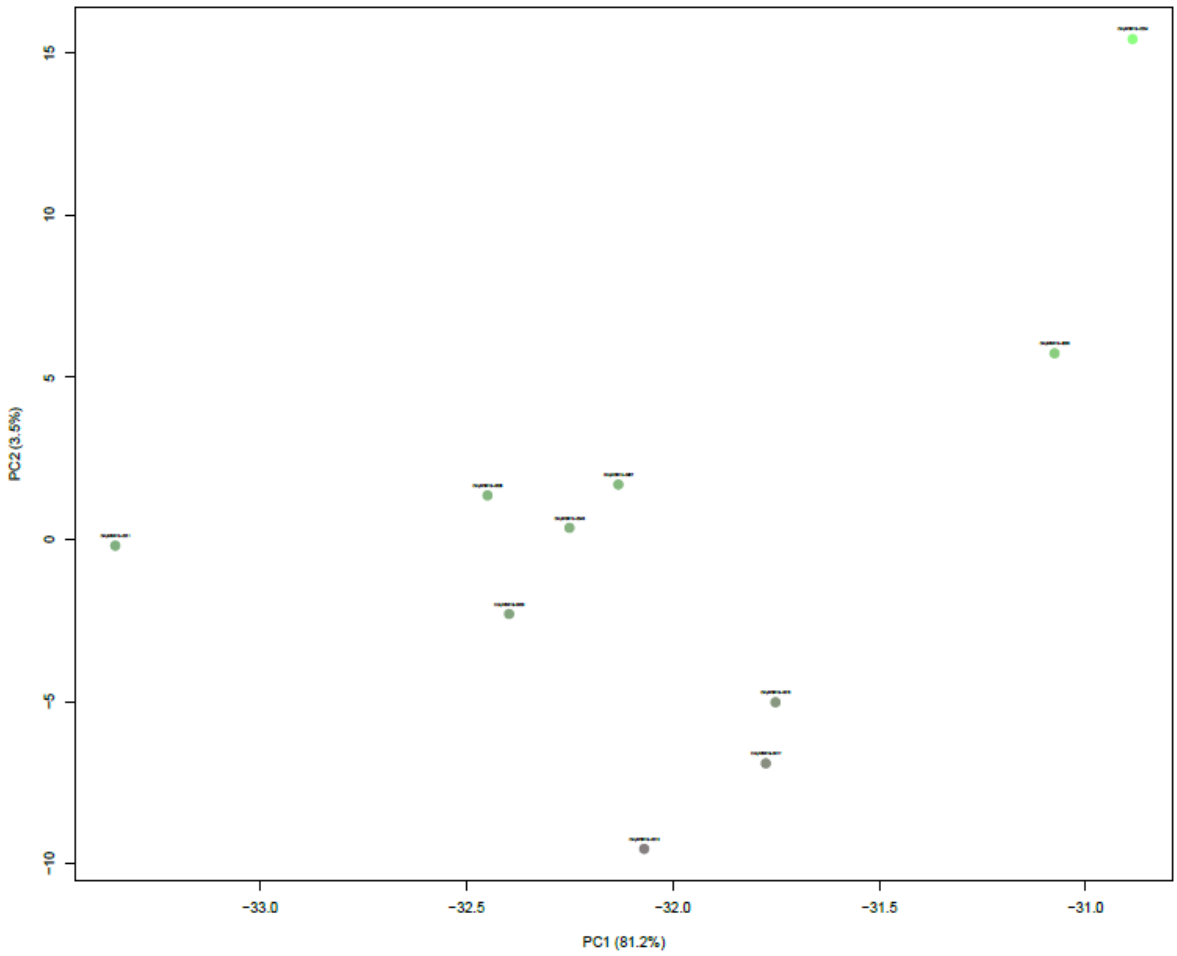
_singleton : Outlier markers that have both low mean MAF and Ho
_fin_fail: Outliers in terms of Ho-He at different MAF levels
_MAF_filt_fail: Minor allele frequency is less than *maf.fz*
_failed_geno: Markers that did not genotype in at least *pop.thres* populations
_badBon: Fail HWE test using p.pop, p.thres, and after Bonferroni correction
_badBY: Fail HWE test using p.pop, p.thres, and after BY correction
_badFDR: Fail HWE test using p.pop, p.thres, and after FDR correction
_badHWE: Fail HWE test using p.pop, p.thres, no correction.

Plots:

fin_plot: Representation of MAF by HW disequilibrium.
he_vs_ho: Representation of HW disequilibrium.
ind_PCA: First two components of PCA by individual genotypes.
pop_PCA: First two components of PCA by population allele frequencies.

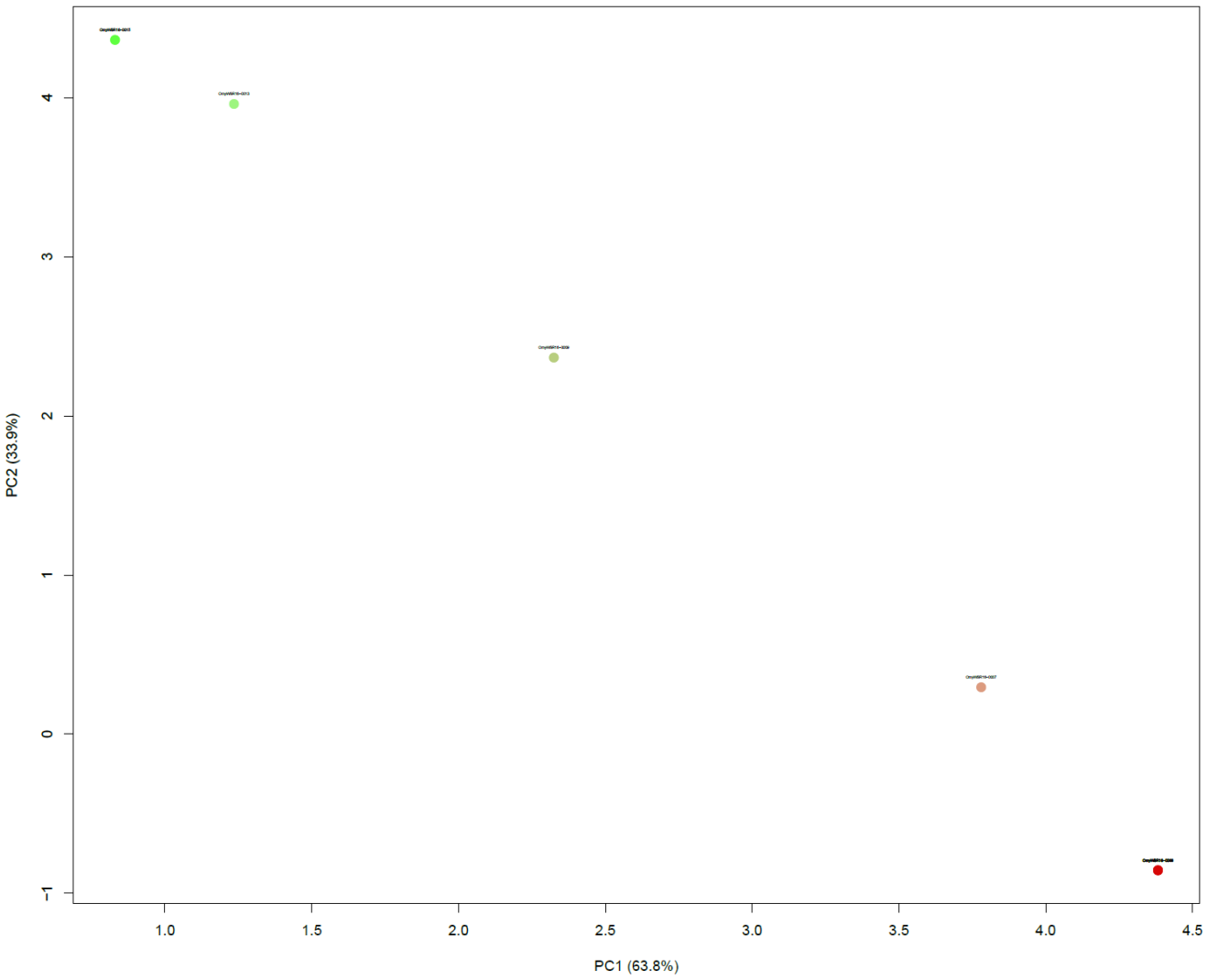
All of this information can help me further filter my loci and, in some cases, individuals. Many times, I will remove problematic loci and re-run *popgen* to see how some of the stats have changed. However, the GT-seq panel is not prone to much error and there is no need to filter it. The information produced has biological relevance and may be of interest.

Looking at the individual PCA plot, I see the 3 geographic localities separated (N=3, N=5, N=2):



Now, I want to focus on the maturation locus-set and see how this PCA changes. Since I'm lazy, I don't need to specify arguments that have standard defaults:

```
> popgen(gpfile="gp_example_mat.gen”)
```



Now individuals are along a gradient of early maturating, mixed, and late maturing. Some individuals are completely overlapping because they have the same genotypes.

I then can open "gp_example_mat_ind_PCA_score.txt" and determine which of maturation markers (alleles) contribute the most to differentiation along PC1.

Using this information, I then may want to run *buildpops* again, sorting individuals into maturation classes as opposed to geographic location. I could address other questions such as how maturating timing might affect gene flow, *etc.*

Simulating Genotypes:

genosim will simulate genotypes (in Genepop format) based on allele frequencies and Hardy-Weinberg disequilibrium. This is useful in our scenario, where we only have 10 individuals and may want to simulate a larger sample size. Even more so, it is a great technique to predict individual genotypes when only have allele frequencies are available from techniques like Pool-sequencing.

Let's increase sample size for our population-informative markers. We can simply use output files from *popgen* to do this.

```
 > genosim(infile="gp_example_pop_freqtable.txt", iter = 100, hwefile =
"gp_example_pop_loc_dis.txt")
```

Note that *genosim* will remove loci that have no variation. We now have "gp_example_pop_freqtable_100_sim.gen" and could run additional analyses on this larger dataset.

Matching loci:

Let's pretend we found an older dataset that has some of the same shared loci as "gp_example_mat.gen." We want to combine these two datasets and run them through *popgen* together, but they have different number of loci that are different order:

> gp_example_mat.gen
Omy_RAD47080$54
Omy_RAD52458$17
Omy_GREB1_05
Omy_GREB1_06
Omy_GREB1_09

> other_mat_file.gen
Omy_GREB1_06
Omy_RAD47080$54
Omy_RAD52458$17
Omy_Old_GREB

We can't simply merge the two files due to this mismatch. *matchloci* solves this mismatch by taking a list of loci as an input and sorting a genepop file in that order. In this case, we want to match the content and order of "other_mat_file.gen." One issue is that "other_mat_file.gen" has Omy_Old_GREB, which is not in "gp_example_mat.gen." *matchloci* deals with this by inserting Omy_Old_GREB into the sorted Genepop file, making all the genotypes missing data. This way, loci can subsequently be filtered out using *sortloci* if needed.

Using the order of:

> other_order.txt

Omy_GREB1_06
Omy_RAD47080$54
Omy_RAD52458$17
Omy_Old_GREB

I run *matchloci*:

```
> matchloci(gpfile="gp_example_mat.gen", loci.file="other_order.txt")
```

and receive: “gp_example_mat_matched.gen” which now matches the same format as “other_mat_file.gen,” and can simply be combined with one another.

III. Conclusions

GPoppin, at its core, can help facilitate many analyses such as outlier detection, Genetic Stock Identification, parentage-based tagging, population-structure, and perform quality control on loci. Since the Genepop format is flexible and widely used, this package is recommended as a starting point for unfiltered genotype data.

Personally, I have used GPoppin and its incorporated PCA techniques to break up the giant MGILCs reporting group into 7 smaller reporting groups. I have used it as a starting point for GSI analyses, preparing the data for ONCOR and GENECLASS2. I have used it for quality control for published RAD datasets. I have used it to merge older datasets with newer datasets to compare genetic data… and much more. There is a lot of utility with this package and it can expedite many operations.