
Genomatic: an R package for DNA fragment analysis project management

Brian J. Knaus
USDA Forest Service
Pacific Northwest Research Station
Corvallis Forest Sciences Laboratory

<http://brianknaus.com>
bknaus at fs dot fed dot us

genomatic 0.0-7
R 2.9
Linux and Windows
January 2010

Acknowledgments:

This package would not have been possible without the efforts of the team involved in the *Purshia tridentata* project at the USDA Forest Service's Pacific Northwest Research Station, Corvallis Forest Sciences lab. In particular, acknowledgments need to be made to Dr. Rich Cronn (USDA PNW), Dr. Matt Horning (USDA PNW), Angie Rodriguez (Oregon State University) and Jane Marler (Oregon State University). This project would not have been possible without their help! I'd also like to thank Dr. Peter Dolan (Oregon State University) for helpful conversation on the R language, computer science, and mathematics in general.

Brian J. Knaus
March 2009

Table of Contents

1	Philosophy, input/output files	4
2	Installation	6
3	Starting the genomatic gui	8
<hr/>		
Pre-genotyping tools		
4	Creating 96-well plate maps.	9
5	Creating ABI submission forms from 96-well plate maps.	11
<hr/>		
Genotype management tools		
6	Scoring genotyper data for input to the genomatic.	12
7	Input of genotypic data to the genomatic.	13
8	Managing reruns – filling holes.	16
<hr/>		
Allele characterization and binning tools		
9	Allele characterization.	18
10	Scoring your data into bins.	22
<hr/>		
Exporting to statistical software		
11	Exporting your data to other software formats. GenAIEx NTSYSpc Create	24
<hr/>		
Appendices		
A	List of Genomatic functions	26
B	List of User supplied files	27
C	File format definitions	28
D	Genomatic flowcharts	33
E	Command line example	39
F	Troubleshooting the genomatic	43

1 Philosophy, input/output files

A general philosophy of this software is to provide modularity and copious output. Modularity allows the user to utilize either the entire package, or more importantly, only certain elements which the user may find useful. The provision of copious output also allows the user to use various steps in this software as inputs or outputs to their preferred software. Throughout this set of programs comma-delimited files (.csv) are used as input and output as much as possible. This file format is considered convenient as it is opened by default by various spreadsheets. It is the author's experience that text files are the common currency among many softwares. Through the provision of comma-delimited text files at many steps throughout the Genomatic process it is believed that the user can use these files for easy evaluation or importing/exporting into other softwares.

In order to manage names of individual samples and populations a convention has been established in this software. Names should be similar to: PUTR_077_01. The underscore ('_') is used to delimit fields within the name. The first field ('PUTR') is a taxon field and is relatively arbitrary but must be present. The second field ('077') is a population code, while the third field ('01') is an individual code. Input to the genomatic frequently only includes the full name and the genomatic parses these names into populations for subsequent sorting.

The Genomatic was created during the summer of 2007 as part of a team involved with a DNA microsatellite project on *Purshia tridentada* (Pursh) DC., a diploid member of the Rose family. The sample involved over 600 individuals which we genotyped at eight loci. Immediately it became obvious that organization would be critical to implementing this project. Furthermore, many of the steps we had been using in the past involved manual 'copying' and 'pasting' which was monotonous and consequently error prone. The Genomatic was created with the specific need of processing a large quantity of diploid microsatellite data, but also with the goal of creating reusable code that could be implemented in other studies.

A graphical user interface is provided with the genomatic. Occasionally the user will need to interact with either the R console or the shell. In order to differentiate these commands the following syntax has been adhered to:

R prompt:

```
> command()
```

Shell prompt:

```
$ command
```

If you enjoy this software please support it by citing it:

Knaus, B.J. In prep. Genomatic: an R package for DNA fragment analysis project management.
[Http://brianknaus.com](http://brianknaus.com).

2 Installation

Obtaining R and its packages

The scripting language 'R' is a free, interpreted language, covered by the GNU public license (<http://www.gnu.org/copyleft/gpl.html>). R is available at <http://www.r-project.org/>. Introductory materials can be found at: <http://www.r-project.org/> and http://oregonstate.edu/~knausb/R_group/R_User.html. Because R is an interpreted language it is easier to learn and implement than lower level languages such as 'C' or 'fortran'. However, the price to pay for more easily interpreted code is slower execution. Large datasets and computationally intensive operations may require noticeable execution time, on the order of minutes or longer. It is the experience of the author that the functions in this package can be executed on the order of minutes (at most, usually much faster) and are therefore very reasonable. Be forewarned that exceptionally large datasets, in samples or markers, may require substantial execution times.

Installing the Genomatic

The genomatic is currently supported for Unix and Windows. I don't have anything against Macs, I just don't use them, or have access to them. The genomatic is written entirely in R, which means 'in theory' it should run on a Mac. If you're interested in using the genomatic on a Mac, send me an e-mail and we can see what we can do.

Command line implementation of the genomatic assumes that the user has a basic understanding of the R language. For more background see the documentation provided at the R website (<http://www.r-project.org/>) or at the author's website (http://oregonstate.edu/~knausb/R_group/R_User.html).

Under **Linux** (I currently use Ubuntu 9 the *Karmic Kolala*) download the tarball to a local directory. Enter R using superuser privileges (e.g, `$ sudo R`). Once in R navigate to the directory containing the tarball and install:

```
> setwd("/media/hdb2") # The directory containing the tarball.
> install.packages("genomatic_0.6-0.tar.gz", repos=NULL)
```

Under **Windows** download the zip file to a local directory. Open R, navigate to the directory containing the zip archive and install (in Windows this can also be accomplished through the drop-down menu):

```
> setwd("c:/") # The directory containing the zip archive.
> install.packages("genomatic_0.6-0.zip", repos=NULL)
```

This can also be accomplished by using the Windows drop-down menu (Packages > Install package(s) from local zip files...).

After installation it may be important to know where the genomatic has been installed. The following R command will tell you where to look:

```
> .find.package("genomatic")
```

or

```
> .libPaths()
```

I've included an R script to install an example to a directory of your choice (where you'll need read and write permissions). This script is the file 'purshia_example_setup.r' in the directory 'genomatic/doc/' within the genomatic installation. Navigate to this directory, open the script and execute the appropriate parts (some are platform specific). You will need to specify a destination directory for the example and make sure this directory exists. This should create a version of the example in your desired directory. This also leaves a copy in the 'genomatic' directory so that you can replace your example directory if you feel it becomes necessary.

In order to use the genomatic open an instance of R and use the command:

```
> library('genomatic')
```

This loads the Genomatic and makes its functions available for use.

Removing the Genomatic

As with all R packages, the Genomatic is installed within the R environment and is installed to a subdirectory of the R installation. In order to remove the package enter the following command:

```
> remove.packages("genomatic")
```

This should remove the Genomatic from your system.

3 Starting the genomatic gui

Functions:

genogui()

The command line implementation of the genomatic will provide the greatest flexibility. A graphical user interface (gui) has also been provided to increase ease of use. The gui is written in the R implementation of tcl/tk, meaning that it should work on most platforms (i.e., Linux, Windows). In order to start the gui, open an instance of R and type the following commands:

```
> library("genomatic")
> genogui()
```

A window should pop-up indicating the initiation of the genomatic (figure 3.1). Two drop-down menus are available. The first, titled 'File' allows the user to exit the genomatic. The second menu, titled 'Genomatic', allows the user to navigate through the genomatic processes.

Many files are accessed and created by the genomatic. Because of this, the specification of input and output directories has been implemented by the genomatic gui and should be performed upon each instance of its use. It is recommended that the user create a directory (folder) specifically for genomatic output to aid project organization. Select the 'Genomatic' drop down menu and under it select 'input/output.' This window allows the user to specify a directory (folder) to start each search for input files and designates a directory (folder) where output files will be written to.

The user should keep in mind that the genomatic gui is simply a front end to command line functions. Important information, such as errors, may only be output to the R console. The gui may be a primary interface, but the user should always check the R console, particularly when unexpected results are encountered.

Subsequent chapters will begin with directions to use the gui which will be followed by a more detailed section addressing the command line.

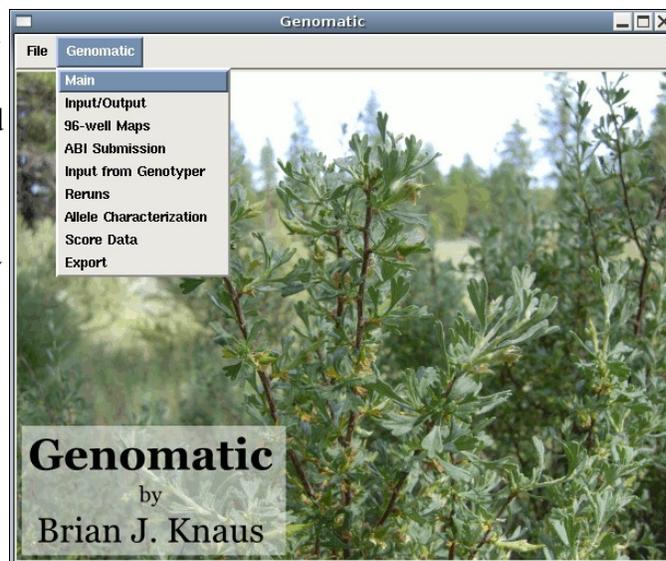


Figure 3.1. Screenshot of the genomatic gui.

4 Creating 96-well plate maps

Functions:

```
pop2indiv  
plater  
plate_write
```

Input files:

population file (see appendix C)

A first step in any population genetic study involves specifying a sample which includes individuals nested within populations. These individuals need to be plated out into 96-well plates for subsequent chemistry and genotyping. This chapter describes how to convert a comma delimited file containing population names and sample sizes into maps of 96-well plates for genotyping. These files are subsequently used to automate generation of ABI submission files (chapter 5).

Graphical user interface:

After specifying input and output directories, select '96-well maps' from the genomatic drop-down menu. Select a population file (see appendix C) and an output prefix, such as 'putr' (an acronym we use for our *Purshia tridentata* project). The function will expand all populations in the population file to include samples 1 to n (as specified in the second column of the file) and arrange them contiguously in 8 by 12 matrices (96-well plates). These matrices will be saved as comma delimited files in the output directory and can be imported into other formats (e.g., spreadsheets or wordprocessor files) and printed to be included in laboratory notebooks.

The genomatic expands populations into samples in a sequential manner (i.e., 1, 2, 3, ... n). If the user wishes, they can manually edit these 96-well plate maps to include non-sequential samples for later use in the genomatic process.

These matrices are also used as input for creating ABI submission forms in subsequent processes. Therefore, creation and editing of these files is encouraged regardless of whether the user plans to directly use them.

Command line:

First we'll need to open R and set the working directory. The working directory is where R will look for files when we tell it to, and will be where files are written to when we write them. The working directory can be set with the following command (an example, note that these directories must exist):

```
> setwd("C:/purshia_example/") # Windows  
> setwd("/home/knausb/purshia_example/") # Linux
```

In the Linux environment the 'working directory' is usually handled by the directory in which files are opened. In Windows the 'working directory' needs to be specified. Recall that Windows uses backwards slashes (\) but because R has its roots in Unix, it uses forward slashes (/). If you 'copy' and 'paste' your directory location out of the Windows Explorer you'll have to change the slashes.

Next we read in a file containing our population data. Examples of all the files you need are included in the `./genomatic/doc` directory which should have been written to a user specified directory when you used the script `'purshia_example_setup.r'`. Remember that R writes over files without asking, which depending on the context may be good or bad.

First we read in the file and save it as the object `'purshia'`. Note that we've included the information that the first line is a header and that it is a comma delimited file. Open this file in a spreadsheet and you will see two columns, one for the population name and a second for the sample size of each population. The second command here uses the sample size to create n individuals for each population where n is the sample size.

```
> purshia <- read.table(file="purshia_pops.csv", header=T,
+                       sep=",")
> p.indiv <- pop2indiv(purshia)
```

Now we organize these samples into an 8 X 12 map (actually a 'matrix'), and finally write these maps to comma delimited files. The second argument in the call to `'plate_write'` (`'putr_plate'`) is a prefix that will be used in the naming of your files (e.g., `putr_plate_1.csv`, `putr_plate_2.csv`). These files are used later in the Genomatic process but may also be useful in creating worksheets to aid the initial plating of samples.

```
> p.plates <- plater(p.indiv)
> plate_write(p.plates, "putr_plate")
```

Note that the function `'plate_write'` parses your data into units of 96-well plates. Therefore you can give it any number of individuals (more or less than 96) and it will give you the correct number of plates. Empty wells are specified by 'NA's.

5 Creating ABI submission forms from 96-well plate maps

Functions:

abi_sub

Input files:

filterset file (appendix C)

96-well plate map (created in chapter 4)

In preparation for genotyping, an ABI genotyping data file needs to be generated. This includes the sample name information as well as the filterset and loci.

Graphical user interface:

After starting the genomatic gui and designating input and output directories (chapter 3) select 'ABI submission' from the genomatic drop-down menu. First, a filterset file needs to be designated, this specifies which fluorophores will be used. See Appendix C for the format definition, as well as the ABI documentation for possible combinations of fluorophores. Second, you'll need a 96-well plate map (see Chapter 4) containing samples to be submitted. This file will be reformatted into the ABI submission form format. Lastly, a prefix to help you identify your project should be included. In our example we've used 'putr.' After inputting this information click on the 'create ABI submission forms' button and a form will be created in the output directory.

The genomatic only supports non-proprietary file formats. The ABI submission forms are therefore created as comma-delimited text. Your genotyping facility may be able to directly utilize this format, or you may have to open the files in a proprietary spreadsheet (i.e., MS Excel) and save the file in the desired format.

Command line:

First read in the filterset file and store it as the object 'fset'. Examples for ABI filterset 'D' and 'G5' are included in the /Genomatic/doc directory. Next read in the plate map, note that the first column contains row names (A, B, ..., H). Lastly, we send the plate, filterset, and a prefix for the outfile (in this case 'purshia_1'). The function abi_sub creates the appropriate information and prints it to a file in the working directory.

```
> fset <- read.table(file="purshia_loci_fsetD.csv", header=T,
+                   sep=",")
> p.plate <- read.table(file="putr_plate_1.csv", header=T,
+                       sep=",", row.names=1)
> abi_sub(p.plate, fset, "purshia_1")
```

The Genomatic currently only supports comma delimited files. The ABI genotyping software may require an MS Excel file. This can easily be accomplished by opening the comma delimited file in MS Excel as saving it as a file of type '.xls.'

6 Scoring Genotyper data for input into the Genomatic

Functions:

Non-Genomatic process

Input files:

None

We score our data using ABI's Genotyper, but you can probably use other software as long as you can get it into our format (see genotypic input file, appendix C). Data is scored such that each locus is a separate 'category,' which results in a table where each 'category' is recorded on a separate row. This results in an individual being spread out over several rows, a problem the genomatic will fix prior to data analysis. The first column of this file should be the file name of the trace (e.g., A10_PUTR_068_01_A10_02.fsa), second column contains the locus (category), third and fourth contain the size of the diploid peaks (in hundredths of base-pairs).

Note that the file name is very specific! It contains information which is delimited by underscores ('_') and will be parsed by the genomatic. The first field of the name (eg., 'A10') indicates the well and is not used subsequently. The second field (e.g., 'PUTR') indicates the project. The third field indicates the population (e.g., '068'). The fourth field indicates the individual (e.g., '02'). Everything after this fourth field is omitted from subsequent processes. If for some reason you are missing this data you can create your own by using the relevant information. If you don't have well information or the end information just use dummy data:

```
something_PUTR_POP_INDIV_something.fsa
```

Homozygous loci may include an 'NA' in column four or have the allele information from column three repeated in column four. If there is data in column three and a 'NA' in column four the genomatic will duplicate the data in column three into column four.

All columns after column four are ignored and may contain anything. The important thing is that all rows are of the same length. If you think you're having trouble with this simply create a dummy row after the rows you want to keep and make sure something occurs in each element of this last row.

Example files can be found in the `/Genomatic/doc/` directory and are named:

```
plate_1_I.txt  
plate_1_II.txt  
plate_5_I.txt  
plate_5_II.txt
```

7 Input of genotypic data to the genomatic

Functions:

```
init_gmtc  
genotyper2genomatic
```

which calls:

```
cat_sorter  
filename2sample  
pop_get  
order_sample  
condense
```

Input files:

locus file (appendix C)
Genomatic file (initiated here, also see appendix C)
Genotypic data input files (see chapter 6 and appendix C)

Once genotypic data has been scored for peak size and formatted appropriately (chapter 6 and appendix C) the data needs to be brought into the genomatic format, a format expected by the genomatic functions. This is facilitated by an initialization function which creates a file called 'gmtc.csv.' Subsequent to this, all data will be added to this file.

Graphical user interface:

At the initiation of a project the user will have to initialize a genomatic file ('_gmtc.csv') before genotypic data can be added to it. This has been facilitated through the 'initialize master file' button. Input for this process is a locus file (see appendix C). This function creates a genomatic file where column one contains populations, column two contains individuals, and all subsequent columns are diploid loci spread over two columns (allele a and b). The 'initialize master file' button creates a file which contains no samples but you can use it as a starting point for adding your genotypic data to.

Once a genomatic file ('_gmtc.csv') has been initialized it can be selected as the 'master file' for addition of more data.

The tcl/tk gui interface utilized by the genomatic only allows the selection of a single file. The genomatic can handle multiple files at this step, in fact it is more recommended to include as many as possible to make your job easier. To facilitate this the user must select an initial file to begin a list of files to be read in using the 'initialize files list' button. The user can then add individual files to the list with the 'add file' button. Repeat this as many times as you have files. Lastly, include a prefix to identify your project, we use 'putr' for our example.

After all your files are added to the list and you've included a project prefix click on the 'convert' button. The genotypic files are converted into genomatic files and concatenated into one file. Multiple instances of the same sample which contain different loci (e.g., different multiplex runs) are also condensed into one record. If we take as an example figure 7.1 we see three instances of a sample.

The screenshot shows a spreadsheet window titled 'isna - OpenOffice.org Calc'. The spreadsheet has columns labeled A through H. Row 1 contains headers: A: population, B: sample, C: FAM_CT1_48a, D: FAM_CT1_48b, E: FAM_CT2_18a, F: FAM_CT2_18b, G: FAM_CT2_21a, H: FAM_CT2_21b. Rows 2-4 contain data with some 'NA' values. Row 5 is empty.

	A	B	C	D	E	F	G	H
1	population	sample	FAM_CT1_48a	FAM_CT1_48b	FAM_CT2_18a	FAM_CT2_18b	FAM_CT2_21a	FAM_CT2_21b
2	PUTR_063	PUTR_063_03	243.13	243.13	NA	NA	144.32	169.11
3	PUTR_063	PUTR_063_03	NA	NA	161.76	161.76	NA	NA
4	PUTR_063	PUTR_063_03	258.76	269.02	NA	NA	159.5	161.63
5								

Figure 7.1. Condensable records.

The first two instances contain different loci and can be condensed by removing 'NAs' but no real data. The third instance cannot and will be dealt with later in managing reruns. Condensing of records starts at the first input file and from top to bottom in each file. This information may be useful in backtracking where samples came from.

Command line:

The file which will hold genotypic data throughout the rest of the genomatic process needs to be initialized before we can add data to it. This is facilitated by using a file which contains the names of the project loci ('purshia_loci.csv') and the function `init_gmtc`.

Initialize 'gmtc.csv'.

```
> setwd(in.dir)
> loci <- read.table('purshia_loci.csv', header=T, sep=",")
> gmtc <- init_gmtc(loci)
> setwd(out.dir)
> write.table(gmtc, file=paste(out.dir, "gmtc.csv", sep="/"),
+             sep=",", row.names=F)
```

Multiple files can be processed at once. This is done by creating a vector of file names and using this to read in each file to a different element of a list.

```
# Read in files.
```

```
> fnames <- c('plate_1_I.txt', 'plate_1_II.txt', 'plate_5_I.txt',  
+             'plate_5_II.txt')  
> setwd(in.dir)  
> purshia.mp <- list()  
> for(i in 1:length(fnames)){  
+   purshia.mp[[i]]<-read.table(fnames[i], header=T, sep="\t")  
+ }
```

Now we're ready to format our data. This is done by loading our master file and using the function 'genotyper2genomatic' to add the new files. Then we use the function 'condense' to combine instances of the same sample which do not contain information on the same loci (figure 7.1).

```
# Read in master file.
```

```
> setwd(out.dir)  
> gmtc <- read.table('gmtc.csv', header=T, sep=",")  
> putr_data <- genotyper2genomatic(gmtc, purshia.mp)  
> putr_data <- condense(putr_data)  
> setwd(out.dir)  
> write.table(putr_data, paste('putr', "_gmtc.csv", sep=""),  
+             sep=",", row.names=F)
```

The function `genotyper2genomatic` calls several functions to help it with its task of converting the Genotyper tables to a format we can deal with. You may never notice them, but I've included some information on them here in case you find an alternate use for them.

The function `cat_sorter` takes Genotyper tables and sorts them so that each row contains a single individual with several loci. Called by the function `genotyper2genomatic`. This function assumes that there is a column called 'Category' which contains a locus identifier.

The function `filename2sample` takes a column of file names from the `fsa` files and extracts the sample name. The expected format of the file name looks like this: 'A10_PUTR_068_01.fsa' where 'A10' is the well, and 'PUTR_068_01' is the sample name (here its population 068 and individual 01 from that population). This function uses the underscore (`_`) to parse this string and keeps the second, third and fourth elements (here they are 'PUTR_068_01' as the sample name).

The function `pop_get` works in a similar way as `filename2sample` does. This function takes as input the sample names (e.g., `PUTR_068_01`) and, using the underscores as delimiters, extracts the population identifier (e.g., `PUTR_068`).

The function `order_sample` takes the `data.frame` and sorts it based on the first column. In the genomic process this column is the sample name.

8 Managing reruns – filling holes

Functions:

```
dup_checker  
dup_replace  
order_sample
```

Input files:

Genomatic file (appendix C)
Genomatic duplicates removed file (manually edited genomatic file)

Throughout the course of a project certain samples may be rerun due to failed reactions or other issues. In a large project it can be somewhat of a task to sort through a large matrix of samples to find these duplicated samples. Here I've created a couple of functions to help automate this task.

Graphical user interface:

The file created in the genomatic import process (chapter 7) may include instances of a sample where each instance includes data for the same locus (i.e., reruns to verify previous results). To manage these, select 'Reruns' from the genomatic drop-down menu. For the data file select a genomatic file which you think has duplicate records and then click on the 'check for reruns' button. You will receive feedback as to whether the genomatic found reruns. If reruns were found a file called 'duplicates_gmtc.csv' will be written to the output directory which only contains the duplicated records.

To manage duplicates, begin by renaming the 'duplicates_gmtc.csv' file, I recommend 'duplicates_removed_gmtc.csv.' Recall that the genomatic will write over files without asking, so renaming the file is a way of protecting it from being accidentally written over (alternatively you could move it to a new directory). Open this file in your favorite spreadsheet and manually condense all duplicated records so that you have one record per sample with only the allele calls you desire. Save this file as a comma-delimited text file. Now return to the genomatic and select your genomatic file which contains duplicates in the 'data file' field, include the 'duplicates_removed_gmtc.csv' file in the 'processed duplicates file' field and include a project prefix and perhaps an identifier that this file will not include duplicates (e.g., 'putr_singles'). Click on the 'remove reruns' button. The genomatic will start at the first record in the 'duplicates_removed_gmtc.csv' file and remove all instances of it from the main data file. It will then insert the manually selected record from the 'duplicates_removed_gmtc.csv' into the main file. This main file, which now contains no duplicates, will be saved as the filename specified in the 'output_prefix' field.

Command line:

The function `dup_checker` searches your dataset for duplicates. The first column of your `data.frame` should be the file's name. The function takes the first sample name and checks for duplicate names in the entire dataset. If a duplicate is found then they are saved in a new `data.frame` which is sorted and returned for you to scrutinize. At the end of execution the function it prints a quick message

telling you whether duplicates were found. If duplicates are found you should save the data.frame as a comma delimited file so you can manually sort through the file and subsequently create a file with no duplicates.

```
# Check for reruns.
```

```
> setwd(out.dir)
> putr_data <- read.table('putr_gmtc.csv', header=T, sep=",")
> putr_dups <- dup_checker(putr_data)
>
> if (is.null(putr_dups) != T){
+   write.table(putr_dups, "duplicates_gmtc.csv", sep=",",
+               row.names=F)
+ }
```

Once we have our file of duplicates we can open it in our favorite spreadsheet and edit it so that the file contains a single instance of each sample. We can then read this file back into the Genomatic and use the function `dup_replace` to remove the duplicates from our data file and add our edited samples. Lastly we order the samples and save them as a comma delimited file.

```
# Process reruns.
```

```
> setwd(out.dir)
> putr_data <- read.table('putr_gmtc.csv', header=T, sep=",")
> putr_dups <- read.table('duplicates_removed_gmtc.csv',
+                         header=T, sep=",")
> putr_data <- dup_replace(putr_data, putr_dups)
> putr_data <- order_sample(putr_data)
> setwd(out.dir)
> write.table(putr_data, paste('putr', "_gmtc.csv", sep=""),
+             sep=",", row.names=F)
```

The file you have edited to remove duplicates can be reused or added to through multiple instances of this process, for example, if you have several rounds of reruns. It is however important to rename this file if you want to keep it because R will write over files without asking.

9 Allele characterization

Functions:

`bin_init`
`allele_process`

which calls:

`allele_char`
`allele_hist`
`bin_by_num`

Input files:

Genomatic file (`_gmtc.csv`)
allele files – located in output directory

A first step in converting mobility-based DNA fragment size data is the creation of bins. Fragment size data consists of an estimate of fragment size in hundredths of a base pair. These estimates need to be grouped into bins which are named for discrete fragment lengths (e.g., 125 bp). This chapter describes a method to categorize fragment size estimates into bins, provide descriptive information about these bins (i.e., standard deviations, ranges), graphical representation of the distribution of fragment size estimates and a graphical representation of bin quality based on range.

Note that mobility based estimates of fragment size do not necessarily reflect the number of nucleotides in a fragment (Applied Biosystems 2004). Molecules of equal nucleotide number but different composition (e.g., percent purines) may differ in mobility base estimates of size. Similarly, identical fragments which are labeled with fluorophores of different molecular weights may migrate at different rates. For this latter reason it is recommended to not change fluorophores during a project. An excellent discussion of caveats of microsatellite analysis is Pasqualotto et al. 2007.

Graphical user interface:

From the 'genomatic' drop-down menu select 'allele characterization.' The first time you characterize alleles for a project you'll need to initialize allele files (appendix C) to describe the bins and subsequently score the data. A simple binning method is used to initialize these files. All the alleles in a project are lined up from smallest to largest and everytime a parameterized gap (default size is 0.3 bp) is found in the data a new bin is scored. First select a genomatic file for the 'data file' field. Next, enter a gap size in the 'minimum gap size' field and click on the 'initialize allele files' button. Allele files will be created for each locus in the output file.

The genomatic binning process is not intended to optimize bins but is instead intended to give the user a good first start. The user should scrutinize these bins and manually adjust them. The genomatic provides tools to help with this. By selecting a genomatic file for the 'data file' field and clicking on the 'characterize alleles' button a number of diagnostics will be generated. First, the locus names will be determined from the header of the genomatic file. These names are used to read in allele files from the output directory (e.g., 'locusname_alleles.csv'). Columns eight and nine from the allele files are used to define bins. The data are binned and summary statistics are generated and added to the allele files

(appendix C). All the alleles are plotted in histograms where each bar is 0.1 bp wide and bins are represented by alternating colors. Lastly a plot of bin width as a function of the number of called alleles for each bin is created to assess bin quality. The latter two plots are created as '.pdf' files and written to the output directory. The allele files are updated and written back to file. If the user has a set of allele files they wish to preserve they should copy them to a new directory (but leave a set in the output directory for the genomatic to read) as the genomatic will write over them without asking.

The allele files are the heart of the binning process. Columns eight and nine are used as the minimum and maximum bin boundary. Every time the 'characterize alleles' button is clicked the data are binned and summary statistics are updated, but columns eight and nine are not manipulated. The 'initialize allele files' uses the minimum and maximum range of each bin as a starting point. The user should open allele files in their favorite spreadsheet and manually edit them as necessary. For example, a mean and standard deviation is included which the user could use to construct 95% confidence intervals for each bin. When these are placed in columns eight and nine and the 'characterize alleles' button is clicked the rest of the allele file is updated. If the user wants to remove a bin simply delete that column (but should probably manipulate the neighboring bins so they include what was in the now deleted bin). To add a bin, insert a row, give the bin a name (column one) and define it by adding values to columns eight and nine. Use the 'characterize alleles' button to update the remaining information.

Along with the scrutiny of the alleles the user is expected to fulfill a few other tasks. Each bin needs a unique name (column one). The genomatic tries to do this via a very simple algorithm, but in order to avoid problems downstream in the genomatic process the user needs to make sure they are unique. This can be done by changing the number or adding a decimal or alphabetic character (i.e., 125a and 125b if the user desires two bins at this location). Next, the user needs to make sure that no bins overlap (columns eight and nine). The genomatic uses greater than or equal to (\geq) and less than or equal to (\leq) statements in the binning process. If two bins are contiguous, one defined with a maximum size of 123.4 bp and a second defined with a minimum of 123.4 bp, it may create ambiguity in the binning process. Simply adding 0.001 to the larger bin could simply avoid this.

Command line:

The first step in characterizing alleles is to initialize files which contain information on the alleles, one file for each locus. In order to bin the data the alleles for each locus are sorted by size. Every time a user parameterized gap in allele size occurs a new bin is called. These files are initialized with 'Nas' in place of summary statistics for each bin. Summary statistics will be added by subsequent functions.

```
# Initialize allele files.
```

```
> setwd(out.dir)
> putr_data <- read.table('putr_gmtc.csv', header=T, sep=",")
> gap <- as.numeric(0.3)
> bin_init(putr_data, gap)
```

Now we read in our data file. The project loci are included in the header (first line) of the data file. We use this to make a vector of locus names which we then use to read in all of the allele files.

```
# Characterize.
```

```
> setwd(out.dir)
> putr_data <- read.table('putr_gmtc.csv', header=T, sep=",")
> loci <- names(putr_data)[seq(3, ncol(putr_data), by=2)]
> loci <- substr(loci, 1, nchar(loci)-1)
>
> setwd(out.dir)
> locus.l <- list()
> for (i in 1:length(loci)){
+   locus.l[[i]]<-read.table(paste(loci[i], '_alleles.csv',
+                                 sep=' '),
+                             header=T, sep=',')
+ }
```

The function `allele_process` calls several other functions to generate graphics of the data and stores them in pdf files as well as a comma delimited file for each of the loci containing summary statistics which will later be used to bin the data. The function requires a data.frame of data, a data.frame (of one column) containing locus names, and a specification of the minimum gap size between bins (in units of base pairs).

```
> setwd(out.dir)
> allele_process(putr_data, locus.l)
```

The file `'locus_characterizations.pdf'` includes a plot of each locus where the range of each called bin as a function of the number of peaks in each bin. A simple linear regression line is also included. The file `'allele_histograms.pdf'` includes histograms of the alleles for each locus. Coloring is based on binning from the last two columns of each respective allele file. These files should be examined to scrutinize binning. It is also important to examine these file if analysis is expected to include a mutation model. Due to issues associated with the estimate of fragment size, alleles may be called as even numbers in one part of a capillary and odd in another part. This can be manually corrected for by the user through changing the name of the allele (the first column of the allele file). The allele files are also populated with summary statistics in this process to aid the user in scrutinizing bins. The last two columns of these files are subsequently used for binning. If the user wishes to add bins, rows can simply be inserted. If the user wishes to remove bins, columns can be deleted. During this process the user needs to make sure that bins do not overlap, are sequential (smallest bin size at the top of each file, largest bin at the bottom) and that all alleles have a bin to be included in.

The user may never need to interact with the following functions which are called by the function `'allele_process'`. I have provided some information here on these functions for the user who

may be interested in modifying the genomic.

The function `bin_flagger` flags boundaries for bins. Bins are defined by sorting the alleles from smallest to largest and searching for gaps of size `gaps`. Whenever a gap is encountered a new bin is called. Bins are first identified by a binary code which marks the beginning of each bin. These bins are subsequently numbered and returned by the function.

The function `allele_char` calculates summary statistics for each allele in a locus. This information can subsequently be used to bin the data. Statistics computed (in base pairs) are the mean size, standard deviation, minimum size included in the bin, maximum size, and the count of samples from the dataset that are in the bin.

The function `bin_by_num` plots bins by the number of samples represented in each bin. The data are plotted by count of samples in the allele (abscissa) by the range (in base pairs) of the bin. A heavy line is plotted at one base pair and thinner lines are plotted at 1.1, 1.2, 1.3, and 1.4 base pairs. It seems intuitive that a bin should not range more than a base pair, however experience suggests that bins that range more than a single base pair are frequently acceptable. This could be due to error in the size calling (how close the peak is to a size standard), among capillary error (capillary quality may induce error), or other unforeseen issues. These plots are intended to help the user assess bin quality and to decide what may be appropriate.

Applied Biosystems. 2004, posting date. Microsatellite analysis on the Applied Biosystems 3130 series genetic analyzers. <http://docs.appliedbiosystems.com/pebi docs/00113901.pdf>.

Pasqualotto, A.C., D.W. Denning and M.J. Anderson. 2007. A cautionary tale: lack of consistency in allele sizes between two laboratories for a published multilocus microsatellite typing system. *Journal of Clinical Microbiology* 45(2): 522-528.

10 Scoring data into bins

Functions:

```
bin_score
  which calls:
  bin_caller
  pop_get
  sample_sorter
```

Input files:

Genomatic file ('_gmtc.csv')
allele files – located in output directory

After bins have been delimited (chapter 10) it is necessary to assign the mobility-based estimates of fragment size (decimal base-pairs) to a bin (an integer). A genomatic file (appendix C) containing allelic data must be read into the genomatic. The names of the loci are extracted from the header and used to read in the allele files (appendix) which contain bin delimitation information. The allele files must exist in the working directory in order for the genomatic to find them. A file containing the original allelic data and the bin calls is written to the output directory. The user can scrutinize this file and make any manual changes they feel is appropriate. The allelic information will be removed in the 'export' process (chapter 12).

Graphical user interface:

In the 'data file' field select a genomatic file containing the data you wish to bin. In the 'output prefix' field input a prefix to identify your project (e.g., 'putr_binned'). Click on the 'score bins' button. Because this is a computensively intensive step this make take on the order of minutes to execute. A file containing the output will be written to the output directory.

Command line:

The function `bin_score` controls the bin scoring process, however it needs some information to do its job. First, it needs data, which we've produced and formatted in previous chapters (chapters 8 & 9). Next we need a table of locus names. The allele characterization process (chapter 9) created a file for each locus containing a characterization for each allele. Again, we get the locus names from the data file header

```
> setwd(out.dir)
> putr_data <- read.table('putr_gmtc.csv', header=T, sep=",")
> loci <- names(putr_data)[seq(3, ncol(putr_data), by=2)]
> loci <- substr(loci, 1, nchar(loci)-1)
```

We give this information to `bin_score` and it scores our data and creates a comma delimited file.

This is an iterative process which will take a bit to execute, in my experience this may be on the order of minutes.

```
> setwd(out.dir)
> bin_score(putr_data, loci,
+           paste('putr', "_scored_gmtc.csv", sep=""))
```

As with any automated scoring process, there may be some calls which are unsatisfactory. It is recommended that this is used as a first draft of your scoring process. There will undoubtedly be some 'difficult alleles' to call in your dataset which will require human intervention to score appropriately.

11 Exporting your data to other software formats

Functions:

```
genobins  
genomatic2genalex  
genomatic2ntsys
```

Input files:

Genomatic file containing allelic and binned data (chapter 11)

The final step of the genomatic is to remove the allelic data leaving just the binned data for subsequent analysis by your favorite statistical package. This can be accomplished via a simple data matrix. This matrix can be used as input for the software 'Create' (Coombs et al. 2008). I've also included support for a couple of my favorite softwares (GenAlEx and NTsyspc).

Graphical user interface:

In the 'data file' field select a genomatic file containing allelic and binned data (created in chapter 11). In the 'output prefix' field select a name which identifies your project (e.g., 'putr'). Click on the button of the output format you desire. A descriptive extension will be added to your prefix to identify your choice of output (e.g., 'putr_matrix.csv').

Command line:

The Genomatic was not created to analyze data, there exist numerous softwares that accomplish this. However, we do need to get the data in a format that these softwares accept. GenAlEx v6 (Peakall and Smouse 2006) is not only a very nice analysis package but it also provides export of data into numerous formats for other softwares. Therefore it was decided this would be a good format for the Genomatic to export to as it could not only provide analysis but also serve as an intermediary to other packages. I personally enjoy NTsyspc, so I've included an export option for this format. Lastly I've included a 'matrix' format which can be used by the software 'Create' (Coombs et al. 2008) to export into many software packages.

First we input our data and remove the allelic information, leaving just the binned data. This is accomplished with the function `genobins`.

```
> setwd(out.dir)  
> putr_data <- read.table('putr_gmtc.csv', header=T, sep=",")  
> putr_data <- genobins(putr_data)
```

Now we can export to GenAlEx format:

```
> setwd(out.dir)
> genomatic2genalex(putr_data, 'putr')
```

We can export to NTSys format:

```
> setwd(out.dir)
> genomatic2ntsys(putr_data, 'putr')
```

Or we can export to matrix format:

```
> setwd(out.dir)
> write.table(putr_data, paste('putr', "_matrix.csv", sep=''),
+             sep=" ", row.names=F)
```

Coombs, J.A., B.H. Letcher and K.H. Nislow. 2008. Create: a software to create input files from diploid genotypic data for 52 genetic software programs. *Molecular Ecology Resources* 8(3): 578-580.

Peakall, R., Smouse, P.E., 2006. GENALEX 6: genetic analysis in Excel. Population genetic software for teaching and research. *Molecular Ecology Notes* 6: 288-295.

Appendix A

Genomatic Functions

This appendix lists the functions included in the genomatic package. This list is also present in the 'INDEX' file.

<code>abi_sub</code>	Creates ABI submission forms
<code>allele_char</code>	Characterizes bins
<code>allele_hist</code>	Creates histograms of binned alleles
<code>allele_process</code>	Processes alleles into bins
<code>bin_by_num</code>	Graphical output of bin quality
<code>bin_caller</code>	Bins peaks
<code>bin_init</code>	Initializes allele files
<code>bin_score</code>	Automates bin scoring
<code>cat_sorter</code>	Sorts categories to their sample
<code>condense</code>	Merges samples with non-redundant data.
<code>dup_checker</code>	Checks for duplicate samples
<code>dup_replace</code>	Replaces duplicate samples with a single sample
<code>filename2sample</code>	Extracts sample names from file names
<code>genobins</code>	Remove peaks from files and leave bins
<code>genogui</code>	Graphical user interface.
<code>genomatic2genalex</code>	Converts data to GenALEX format
<code>genomatic2ntsys</code>	Converts data to NTSYSpc format
<code>genotyper2genomatic</code>	Import genotyper tables
<code>init_gmtc</code>	Initialize genomatic data file.
<code>order_sample</code>	Order samples alphanumerically
<code>plater</code>	Organizes individual names into 96-well plates
<code>plate_write</code>	Write 96-well maps
<code>pop2indiv</code>	Population Name to Individual Name Conversion
<code>pop_get</code>	Extracts the population name from a sample's name
<code>sample_sorter</code>	Sorts data from separate lists into a data.frame

24 total functions

Appendix B

User supplied files

In order to complete the entire Genomatic process several files need to be provided by the user. Examples of these files are included in the `/genomatic/doc/` directory. Here is a list of these files.

<code>purshia_pops.csv</code>	Site name and sample size.
<code>purshia_loci_fsetD.csv</code>	Filterset and locus information.
<code>purshia_loci_fsetG5.csv</code>	
<code>plate_1_I.txt</code>	Exported Genotyper tables.
<code>plate_1_II.txt</code>	These four files include two plates (1 & 5)
<code>plate_5_I.txt</code>	with two different multiplexes (I & II).
<code>plate_5_II.txt</code>	
<code>purshia_loci.csv</code>	Lists each locus.

An example script containing the commands in the manual can be found in the `/genomatic/doc/` directory called `purshia_example.r`. When you run this script it will create numerous output files. In the example I have chosen to place these files in the 'output' directory, this directory must exist or an error will be generated (the script `purshia_example_setup.r` will create an output directory). Keep in mind that R will overwrite files without asking, this can be both good and bad depending on the circumstances.

Appendix C

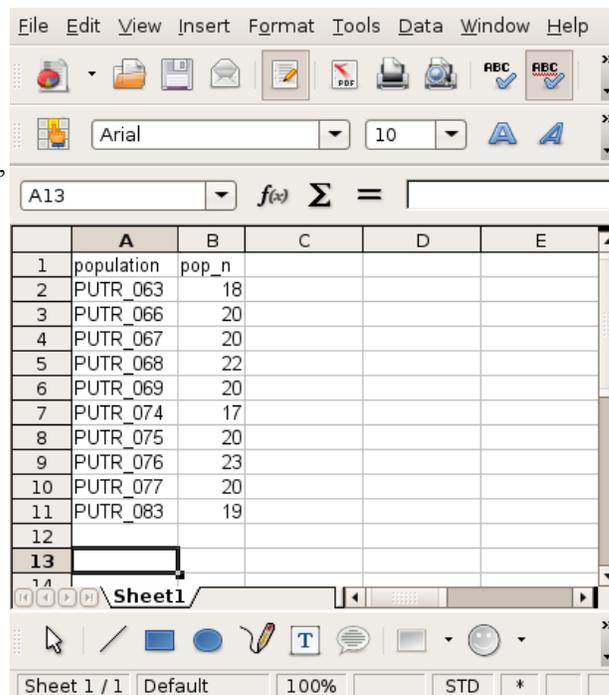
Input file formats

Below are the definitions for file formats included in the genomatic. I recommend managing files in your favorite spreadsheet. These examples were prepared using OpenOffice.

Population file. A population file is used in the process of creating 96-well plate maps. This also helps establish the naming convention for samples used throughout the genomatic process.

File checklist:

- Comma delimited text.
- First row is a header.
- First column is a population name. This name includes a taxon or project identifier (e.g., 'PUTR') as well as a unique population number delimited by an underscore ('_'). (The underscore is important and is used in subsequent processes.)
- Second column is the number of individuals in each population.
- All rows must be of the same length.



The screenshot shows an OpenOffice spreadsheet window with a menu bar (File, Edit, View, Insert, Format, Tools, Data, Window, Help) and a toolbar. The spreadsheet has a grid with columns A through E and rows 1 through 13. The data is as follows:

	A	B	C	D	E
1	population	pop_n			
2	PUTR_063	18			
3	PUTR_066	20			
4	PUTR_067	20			
5	PUTR_068	22			
6	PUTR_069	20			
7	PUTR_074	17			
8	PUTR_075	20			
9	PUTR_076	23			
10	PUTR_077	20			
11	PUTR_083	19			
12					
13					

Figure C.1. Population file viewed in OpenOffice.

Filterset file. The filterset file is used in the preparation of files for submission for ABI genotyping.

File checklist:

- Comma delimited text.
- First row is a header.
- Columns 1-3 describe information specific to the ABI file format.
- Column 4 describes the loci included in each color (fluorophore).
- All rows must be of the same length.

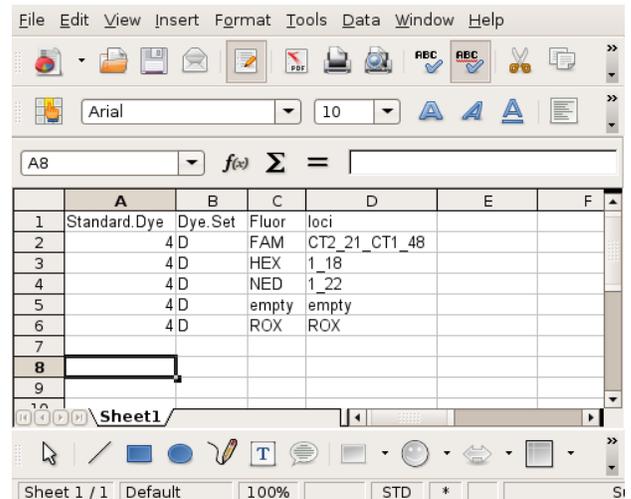


Figure C.2. Filterset file viewed in OpenOffice.

Locus file. The locus file is used to initialize a genomatic file. A locus file includes all loci used in a project, as opposed to a filterset file which only includes the loci in a single filterset.

File checklist:

- Comma delimited text.
- First row is a header.
- First column is a unique identifier for each locus.
- All rows must be of the same length.

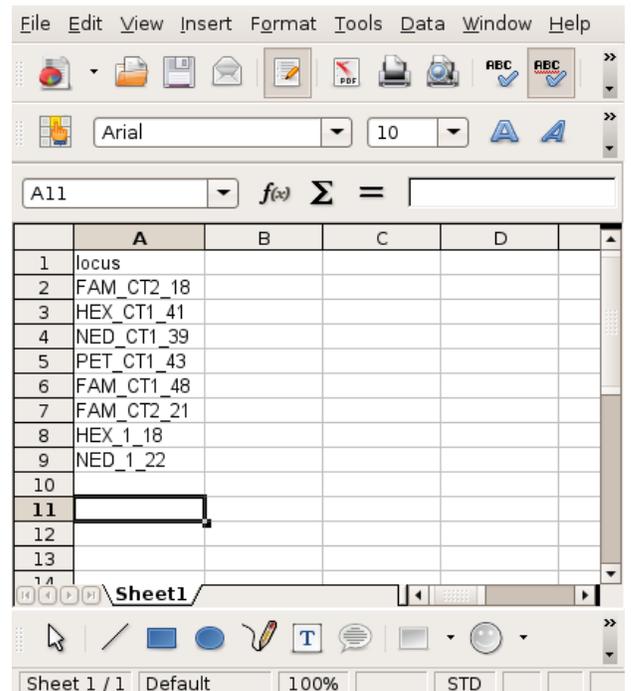


Figure C.3. Locus file viewed in OpenOffice

Genotypic input. This is the format of files produced by genotype calling software (e.g., ABI's genotyper or genemapper).

File checklist:

- Tab ('\t') delimited text.
- First row is a header.
- First column is a unique identifier for each chromatogram file. This field includes the well, taxon or project identifier, sample number and other information, all delimited by underscores ('_').
- The second column identifies the locus.
- The third and fourth columns includes the fragment size in base pairs.
- Empty cells in columns 3 and 4 are interpreted as Nas.
- Samples which contain a value for column 3 but no value for column 4 are scored as homozygotes.
- All other columns are ignored.
- All rows must be of the same length.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	File Name	Category	Peak 1	Peak 2	Lane	Dye	Sample Info	Sample Comment					
2	A10_PUTR_068_01_A10_02.fsa	FAM_CT1_48	242.92	246.8	2	B	FAM	A10_PUTR-068-01_FAM_CT1-48_CT2-21					
3	A10_PUTR_068_01_A10_02.fsa	FAM_CT2_21	144.35	169.21	2	B	FAM	A10_PUTR-068-01_FAM_CT1-48_CT2-21					
4	A10_PUTR_068_01_A10_02.fsa	HEX_1_18	188.36	188.36	2	G	HEX	A10_PUTR-068-01_HEX_1-18					
5	A10_PUTR_068_01_A10_02.fsa	NED_1_22	192.38	220.83	2	Y	NED	A10_PUTR-068-01_NED_1-22					
6	A11_PUTR_068_09_A11_01.fsa	FAM_CT1_48	240.95	242.8	1	B	FAM	A11_PUTR-068-09_FAM_CT1-48_CT2-21					
7	A11_PUTR_068_09_A11_01.fsa	FAM_CT2_21	146.4	154.99	1	B	FAM	A11_PUTR-068-09_FAM_CT1-48_CT2-21					
8	A11_PUTR_068_09_A11_01.fsa	HEX_1_18	184.25	188.22	1	G	HEX	A11_PUTR-068-09_HEX_1-18					
9	A11_PUTR_068_09_A11_01.fsa	NED_1_22	192.41	192.41	1	Y	NED	A11_PUTR-068-09_NED_1-22					
10	A12_PUTR_068_17_A12_02.fsa	FAM_CT1_48	242.86	246.89	2	B	FAM	A12_PUTR-068-17_FAM_CT1-48_CT2-21					
11	A12_PUTR_068_17_A12_02.fsa	FAM_CT2_21	144.35	146.23	2	B	FAM	A12_PUTR-068-17_FAM_CT1-48_CT2-21					
12	A12_PUTR_068_17_A12_02.fsa	HEX_1_18	194.26	194.26	2	G	HEX	A12_PUTR-068-17_HEX_1-18					
13	A12_PUTR_068_17_A12_02.fsa	NED_1_22			2	Y	NED	A12_PUTR-068-17_NED_1-22					
14	A1_PUTR_063_01_A01_01.fsa	FAM_CT1_48			1	B	FAM	A1_PUTR-063-01_FAM_CT1-48_CT2-21					
15	A1_PUTR_063_01_A01_01.fsa	FAM_CT2_21			1	B	FAM	A1_PUTR-063-01_FAM_CT1-48_CT2-21					
16	A1_PUTR_063_01_A01_01.fsa	HEX_1_18			1	G	HEX	A1_PUTR-063-01_HEX_1-18					
17	A1_PUTR_063_01_A01_01.fsa	NED_1_22			1	Y	NED	A1_PUTR-063-01_NED_1-22					
18	A2_PUTR_063_09_A02_02.fsa	FAM_CT1_48	243.05	246.99	2	B	FAM	A2_PUTR-063-09_FAM_CT1-48_CT2-21					
19	A2_PUTR_063_09_A02_02.fsa	FAM_CT2_21	165.49	169.19	2	B	FAM	A2_PUTR-063-09_FAM_CT1-48_CT2-21					
20	A2_PUTR_063_09_A02_02.fsa	HEX_1_18	194.43	194.43	2	G	HEX	A2_PUTR-063-09_HEX_1-18					
21	A2_PUTR_063_09_A02_02.fsa	NED_1_22	192.54	192.54	2	Y	NED	A2_PUTR-063-09_NED_1-22					
22	A3_PUTR_063_17_A03_01.fsa	FAM_CT1_48	242.86	246.91	1	B	FAM	A3_PUTR-063-17_FAM_CT1-48_CT2-21					
23	A3_PUTR_063_17_A03_01.fsa	FAM_CT2_21		179	184.97	1	B	FAM	A3_PUTR-063-17_FAM_CT1-48_CT2-21				
24	A3_PUTR_063_17_A03_01.fsa	HEX_1_18	186.5	188.48	1	G	HEX	A3_PUTR-063-17_HEX_1-18					
25	A3_PUTR_063_17_A03_01.fsa	NED_1_22	192.35	194.33	1	Y	NED	A3_PUTR-063-17_NED_1-22					
26	A4_PUTR_066_01_A04_02.fsa	FAM_CT1_48	239.02	247.06	2	B	FAM	A4_PUTR-066-01_FAM_CT1-48_CT2-21					
27	A4_PUTR_066_01_A04_02.fsa	FAM_CT2_21	169.17	177.15	2	B	FAM	A4_PUTR-066-01_FAM_CT1-48_CT2-21					
28	A4_PUTR_066_01_A04_02.fsa	HEX_1_18	191.93	198.08	2	G	HEX	A4_PUTR-066-01_HEX_1-18					
29	A4_PUTR_066_01_A04_02.fsa	NED_1_22	196.3	196.3	2	Y	NED	A4_PUTR-066-01_NED_1-22					
30	A5_PUTR_066_09_A05_01.fsa	FAM_CT1_48	247.28	247.28	1	B	FAM	A5_PUTR-066-09_FAM_CT1-48_CT2-21					
31	A5_PUTR_066_09_A05_01.fsa	FAM_CT2_21	156.61	163.04	1	B	FAM	A5_PUTR-066-09_FAM_CT1-48_CT2-21					
32	A5_PUTR_066_09_A05_01.fsa	HEX_1_18	184.86	188.52	1	G	HEX	A5_PUTR-066-09_HEX_1-18					
33	A5_PUTR_066_09_A05_01.fsa	NED_1_22	192.33	199.77	1	Y	NED	A5_PUTR-066-09_NED_1-22					
34	A6_PUTR_066_17_A06_02.fsa	FAM_CT1_48	226.93	246.92	2	B	FAM	A6_PUTR-066-17_FAM_CT1-48_CT2-21					
35	A6_PUTR_066_17_A06_02.fsa	FAM_CT2_21	155.03	155.03	2	B	FAM	A6_PUTR-066-17_FAM_CT1-48_CT2-21					
36	A6_PUTR_066_17_A06_02.fsa	HEX_1_18	196.22	196.22	2	G	HEX	A6_PUTR-066-17_HEX_1-18					
37	A6_PUTR_066_17_A06_02.fsa	NED_1_22	192.43	192.43	2	Y	NED	A6_PUTR-066-17_NED_1-22					
38	A7_PUTR_067_01_A07_01.fsa	FAM_CT1_48	244.86	248.93	1	B	FAM	A7_PUTR-067-01_FAM_CT1-48_CT2-21					
39	A7_PUTR_067_01_A07_01.fsa	FAM_CT2_21	175.08	182.86	1	B	FAM	A7_PUTR-067-01_FAM_CT1-48_CT2-21					

Figure C.4. Genotypic input file viewed in OpenOffice.

Genomatic file (`_gmtc.csv`). This is the standardized file format for processing reruns and binning alleles.

File checklist:

- Comma delimited text.
- First row is a header.
- First column is a unique identifier for each population.
- Second column is a unique identifier for each sample. This entry includes a taxon or project identifier (e.g., 'PUTR'), a population identifier and a sample number all delimited by an underscore ('_').
- All rows must be of the same length.

The screenshot shows a spreadsheet titled 'putr_gmtc - OpenOffice.org Calc'. The data is organized into columns labeled A through M. Column A is 'population' and column B is 'sample'. Columns C through M contain numerical values for various genetic markers, with 'NA' indicating missing data. The rows represent individual samples, grouped by population (e.g., PUTR_063, PUTR_066).

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	population	sample	FAM_CT1_48a	FAM_CT1_48b	FAM_CT2_18a	FAM_CT2_18b	FAM_CT2_21a	FAM_CT2_21b	HEX_1_18a	HEX_1_18b	HEX_CT1_41a	HEX_CT1_41b	NED_1_22a
2	PUTR_063	PUTR_063_01	NA	NA	161.81	161.81	NA	NA	NA	NA	221.87	221.87	NA
3	PUTR_063	PUTR_063_02	NA	NA	161.83	177.72	150.74	181.01	190.5	190.5	176.81	229.44	192.47
4	PUTR_063	PUTR_063_03	243.13	243.13	161.76	161.76	144.32	169.11	180.66	180.66	176.69	196.19	192.36
5	PUTR_063	PUTR_063_04	242.91	247.02	175.7	185.45	150.9	156.88	188.53	188.53	176.7	198.36	200
6	PUTR_063	PUTR_063_05	NA	NA	181.51	181.51	NA	NA	NA	NA	198.25	202.1	NA
7	PUTR_063	PUTR_063_06	242.91	265.04	161.8	173.72	165.4	165.4	188.62	194.36	196.22	204.12	196.42
8	PUTR_063	PUTR_063_07	NA	NA	163.74	171.72	NA	NA	NA	NA	176.63	196.21	192.47
9	PUTR_063	PUTR_063_08	247.02	248.9	159.91	180.8	155.32	175.24	NA	NA	208.01	218.71	192.5
10	PUTR_063	PUTR_063_09	243.05	246.99	161.87	173.81	165.49	169.19	194.43	194.43	196.67	196.67	192.54
11	PUTR_063	PUTR_063_10	247.09	247.09	180.82	180.82	155.07	175.16	178.18	178.18	176.7	176.7	202.51
12	PUTR_063	PUTR_063_11	247.02	248.94	159.92	161.74	175.21	181.02	190.39	196.28	210.92	216.75	192.41
13	PUTR_063	PUTR_063_12	NA	NA	159.92	161.75	NA	NA	NA	NA	202.17	218.72	NA
14	PUTR_063	PUTR_063_13	246.75	246.75	171.71	179.6	169.24	183.12	188.59	188.59	176.67	198.08	192.43
15	PUTR_063	PUTR_063_14	243.12	247.01	163.7	181.49	153.12	173.24	180.77	195.39	176.55	194.06	202.24
16	PUTR_063	PUTR_063_15	240.72	243	159.83	161.7	150.74	179.01	186.55	194.28	176.58	203.85	192.45
17	PUTR_063	PUTR_063_16	247.07	248.92	161.79	163.74	173.21	185.08	186.67	188.64	209.78	215.66	192.49
18	PUTR_063	PUTR_063_17	242.86	246.91	161.75	161.75	179	184.97	186.5	188.48	NA	NA	192.35
19	PUTR_063	PUTR_063_18	242.81	246.75	159.92	173.66	165.44	175.14	192.18	192.18	216.61	217.89	196.32
20	PUTR_063	PUTR_063_19	242.87	250.87	171.68	173.63	169.12	184.9	182.28	188.44	186.17	217.84	192.28
21	PUTR_063	PUTR_063_20	242.72	246.91	175.66	181.53	150.75	152.81	189.97	199.69	196.16	223.8	NA
22	PUTR_066	PUTR_066_01	239.02	247.06	180.74	180.74	169.17	177.15	191.93	198.08	200.16	206.14	196.3
23	PUTR_066	PUTR_066_02	246.76	250.9	159.75	180.79	144.31	159.28	170.64	191.42	192.29	200.08	192.41
24	PUTR_066	PUTR_066_03	244.84	246.78	163.68	179.49	167.05	177.05	180.12	186.32	200	223.78	192.33
25	PUTR_066	PUTR_066_04	242.84	246.97	159.66	175.63	146.29	175.08	182.1	191.38	195.99	200	192.37
26	PUTR_066	PUTR_066_05	226.9	248.97	167.67	175.58	175.2	180.94	194.01	196.19	209.91	223.78	192.29
27	PUTR_066	PUTR_066_06	238.9	238.9	161.79	163.67	163.15	173.03	172.01	184.78	198.14	200.09	NA
28	PUTR_066	PUTR_066_07	242.67	250.97	161.89	177.6	171.08	173.13	182.08	186.31	204.07	235.38	192.31
29	PUTR_066	PUTR_066_08	226.74	238.69	161.63	175.51	157.32	163.45	188.59	194.27	195.98	203.98	192.43
30	PUTR_066	PUTR_066_09	247.28	247.28	177.46	177.46	156.61	163.04	184.86	188.52	NA	NA	192.33
31	PUTR_066	PUTR_066_10	238.87	242.81	165.73	177.59	146.4	173.15	200	200	196.25	215.66	192.37
32	PUTR_066	PUTR_066_11	226.85	246.77	161.7	177.59	163.38	180.92	172.29	172.29	197.89	197.89	192.34
33	PUTR_066	PUTR_066_12	243.2	243.2	161.69	187.32	146.6	150.34	186.41	186.41	194.21	201.95	192.29
34	PUTR_066	PUTR_066_13	243.08	247.03	175.58	177.62	150.84	163.32	NA	NA	196.11	198.06	192.29
35	PUTR_066	PUTR_066_14	242.54	250.77	161.73	180.65	171.13	173.44	NA	NA	168.83	209.8	192.33
36	PUTR_066	PUTR_066_15	244.88	246.79	161.9	161.9	153.17	157.13	170.61	194.18	176.32	195.72	192.23
37	PUTR_066	PUTR_066_16	242.89	252.93	161.64	177.46	163.26	175.16	NA	NA	176.42	194.13	192.37
38	PUTR_066	PUTR_066_17	226.93	246.92	163.79	175.69	155.03	155.03	196.22	196.22	168.56	223.75	192.43
39	PUTR_066	PUTR_066_18	226.59	246.94	163.73	181.55	166.92	169.09	194.39	194.39	194.27	196.24	192.18

Figure C.5. Genomatic data file (`_gmtc.csv`) viewed in OpenOffice.

Allele file (_alleles.csv). This file contains the alleles called for a locus.

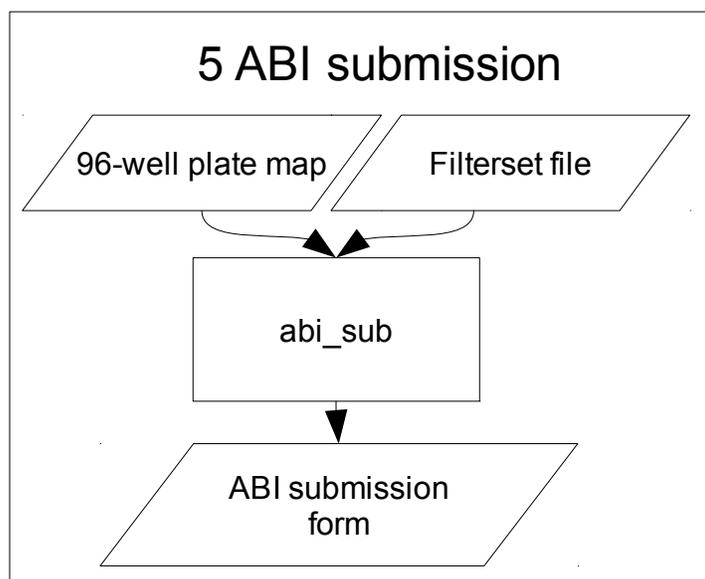
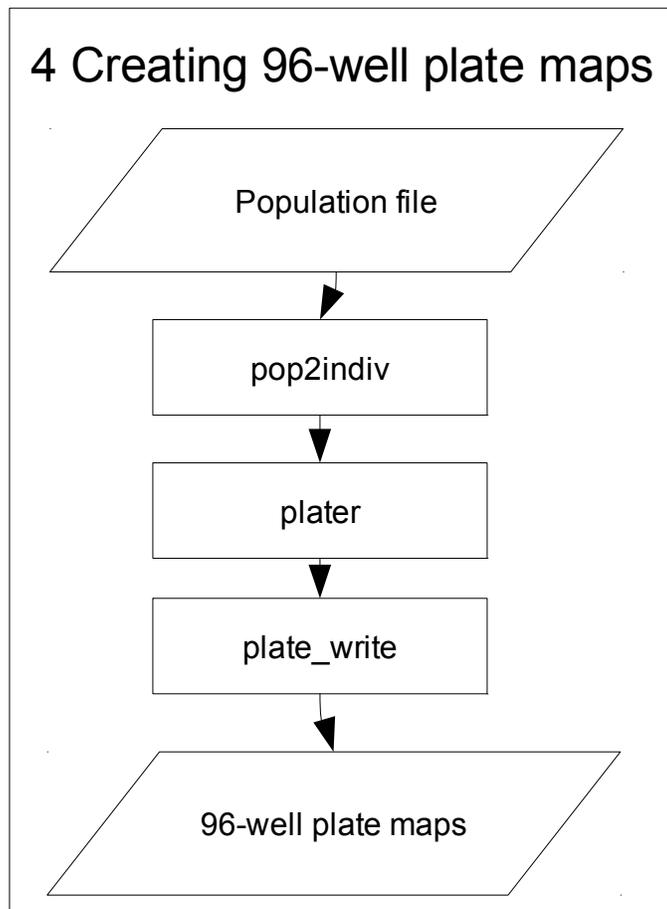
File checklist:

- Comma delimited text.
- First row is a header.
- First column is a name for each allele derived from its molecular weight (check for duplicates).
- Second column is the mean molecular weight for each allele.
- Third column is the standard deviation for each allele.
- Fourth and fifth columns are the minimum and maximum size for each allele.
- Sixth column is the number of alleles in each bin.
- Seventh column is the range for each bin.
- Eighth and ninth columns are used by subsequent processes to bin the alleles. By default these are the minimum and maximum size. The user may manually change the values in these columns to customize binning. For example, means and standard deviations could be used to construct 95% confidence intervals. (Make sure no bins overlap!)
- All rows must be of the same length.

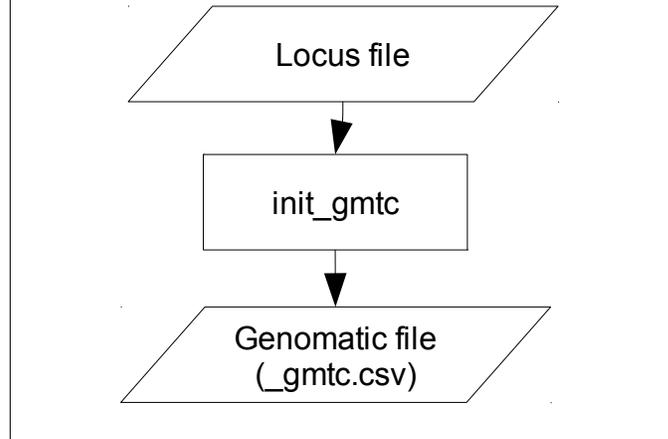
	A	B	C	D	E	F	G	H	I	J
1	allele_name	allele_mu	allele_sd	allele_min	allele_max	allele_n	allele_rge	bin_min	bin_max	
2	225	225.4	NA	225.4	225.4	1	0	225.4	225.4	
3	227	226.85	0.2	226.59	227.21	9	0.62	226.59	227.21	
4	239	238.99	0.21	238.55	239.38	19	0.83	238.55	239.38	
5	241	241.1	0.19	240.68	241.65	56	0.97	240.68	241.65	
6	243	243.04	0.16	242.41	243.63	341	1.22	242.41	243.63	
7	245	244.91	0.3	244.1	245.3	97	1.2	244.1	245.3	
8	247	246.96	0.31	246.18	247.89	235	1.71	246.18	247.89	
9	248	248.3	0.05	248.25	248.37	9	0.12	248.25	248.37	
10	250	249.64	0.86	248.81	251.19	151	2.38	248.81	251.19	
11	252	252.15	0.02	252.12	252.16	3	0.04	252.12	252.16	

Figure C.6. Allele file viewed in OpenOffice.

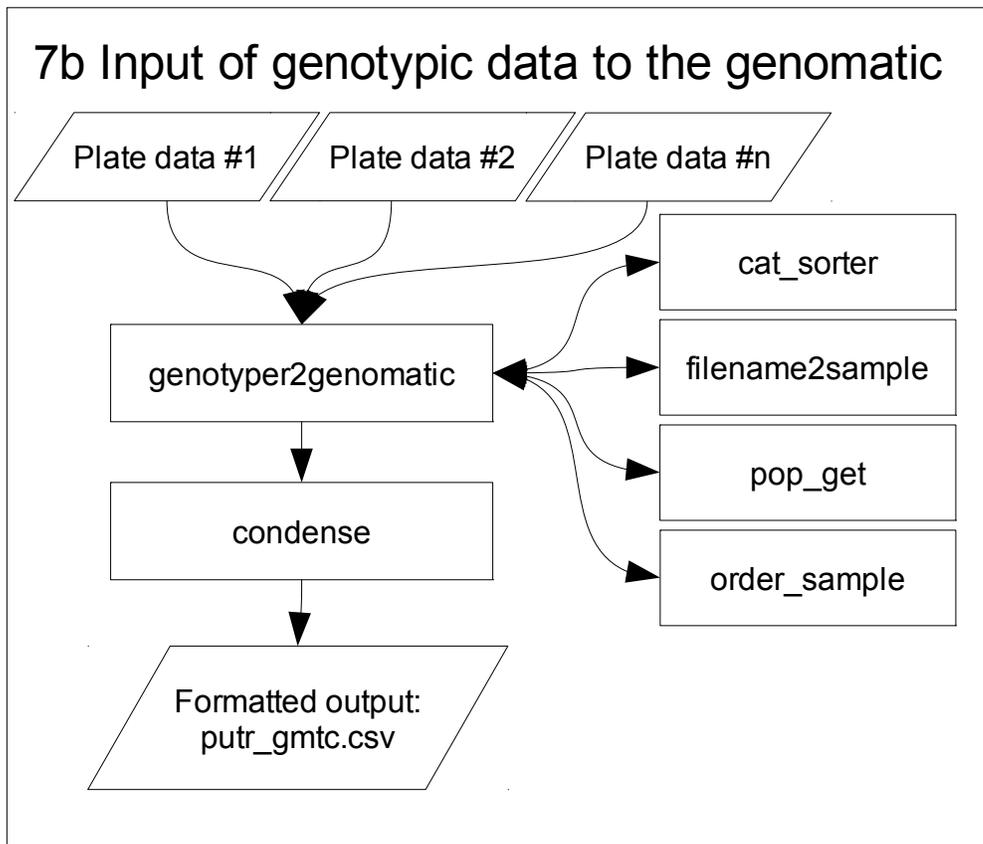
Genomic flowcharts



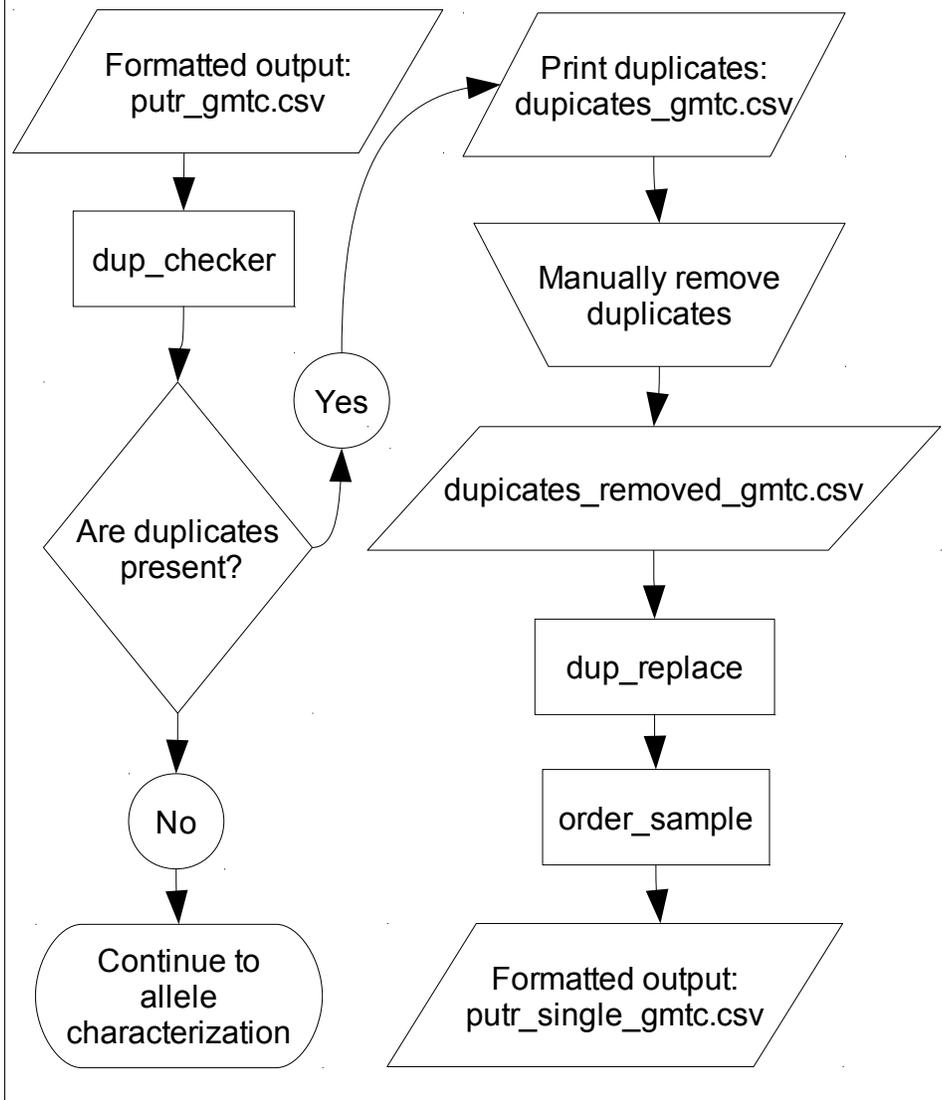
7a Genomatic file initialization



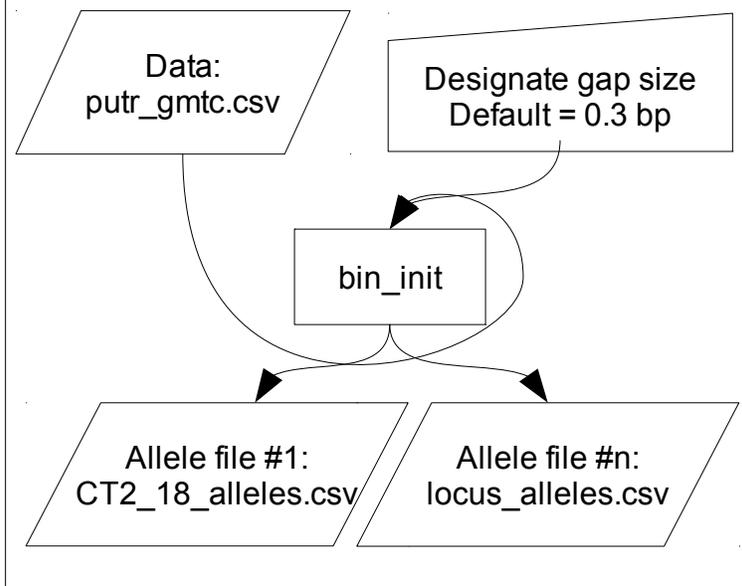
7b Input of genotypic data to the genomatic



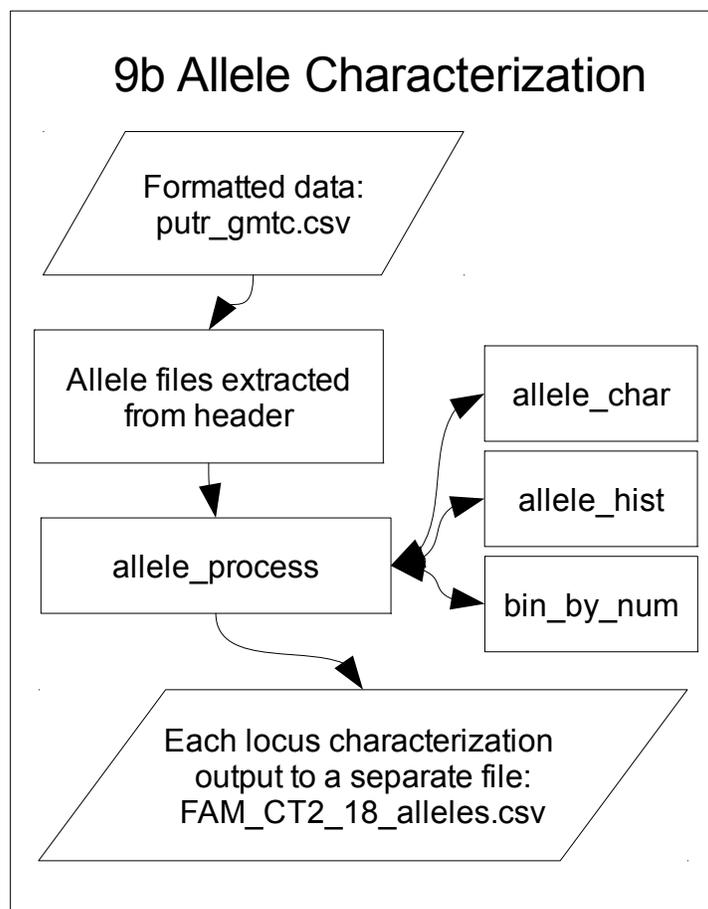
8 Managing Reruns



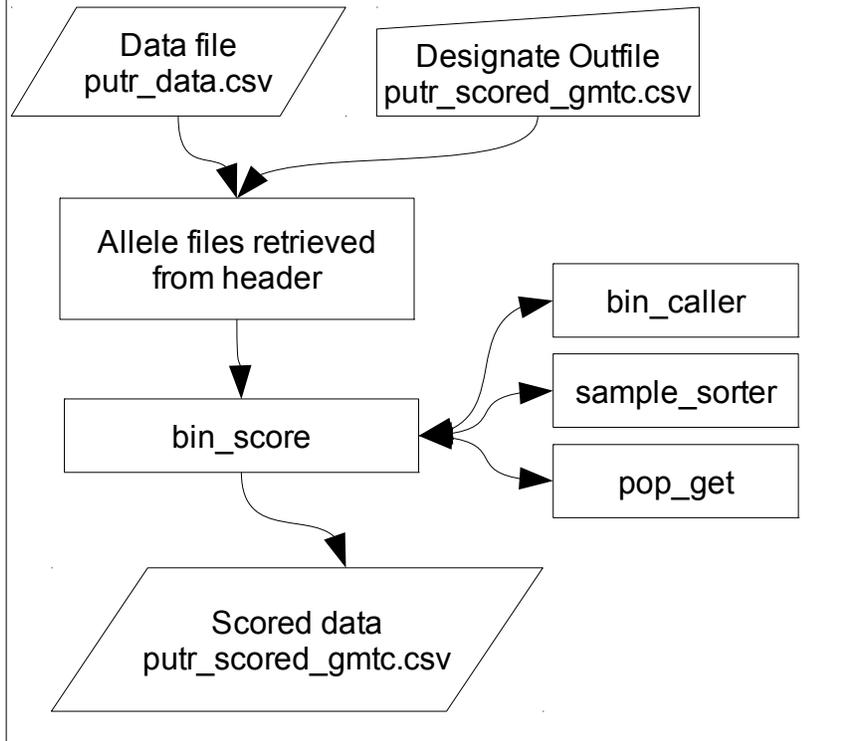
9a Allele file initialization



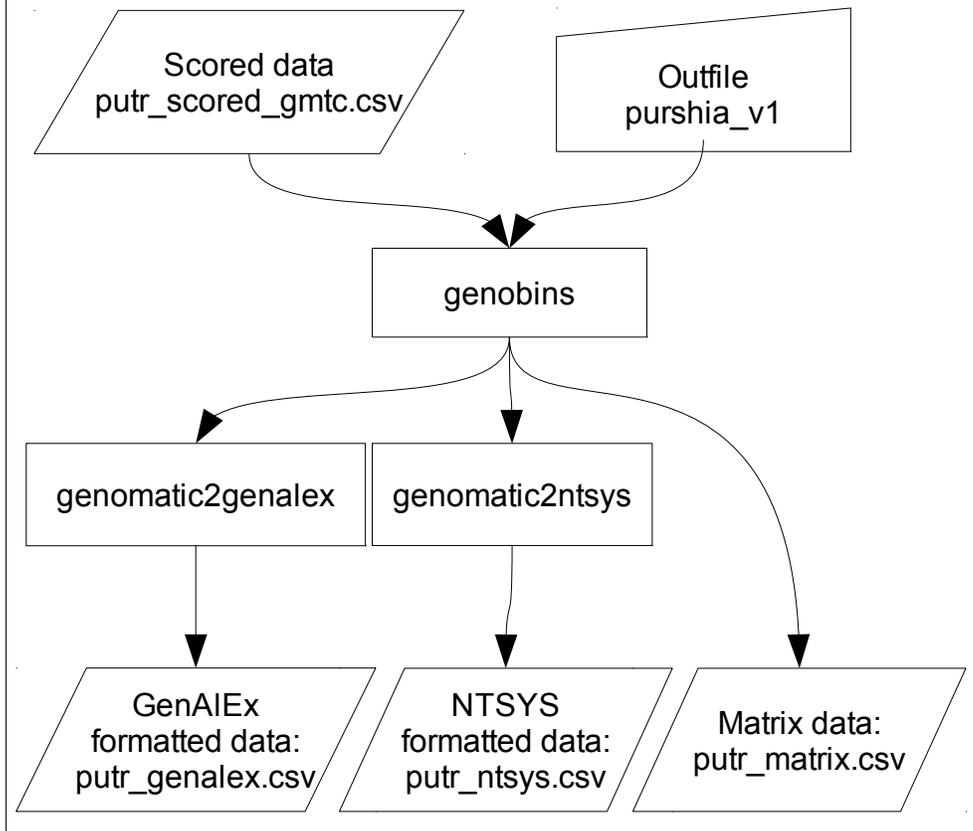
9b Allele Characterization



10 Bin Scoring



11 Export



Appendix E

Command line example

```
# #####  
#  
# Load genomatic library #  
#  
# #####  
  
library(genomatic)  
  
# #####  
#  
# 96-Well Maps. #  
#  
# #####  
  
purshia <- read.table("purshia_pops.csv", header=TRUE, sep=",")  
p.indiv <- pop2indiv(purshia)  
p.plates <- plater(p.indiv)  
setwd("output")  
plate_write(p.plates, "putr_plate")  
  
# #####  
#  
# ABI Submission. #  
#  
# #####  
  
fset <- read.table("purshia_loci_fsetD.csv", header=TRUE, sep=",")  
p.plate <- read.table("output/putr_plate_1.csv", header=TRUE,  
                      sep=",", row.names=1)  
setwd("output")  
abi_sub(p.plate, fset, "putr")  
setwd("../")  
  
# #####  
#  
# Input genotypic data to the Genomatic. #  
#  
# #####  
  
# Initialize genomatic file.  
setwd("output")
```

```

loci <- read.table("../purshia_loci.csv",
                  header=TRUE, sep=",")
gmtc <- init_gmtc(loci)
write.table(gmtc, "gmtc.csv",
           sep=",", row.names=FALSE)

# Read in files.
setwd("../")
purshia.mp <- list()
fnames <- c("plate_1_I.txt", "plate_1_II.txt",
           "plate_5_I.txt", "plate_5_II.txt")
for(i in 1:length(fnames))
{
  purshia.mp[[i]]<-read.table(fnames[i], header=TRUE,
                             sep="\t")
}

# Read in master file
gmtc <- read.table("output/gmtc.csv", header=TRUE, sep=",")
#

putr_data <- genotyper2genomatic(gmtc, purshia.mp)
setwd("output")
putr_data <- condense(putr_data)

write.table(putr_data,
           paste("putr", "_gmtc.csv", sep=""),
           sep=",", row.names=FALSE)

# #####
#
# Managing Reruns. #
#
# #####

putr_data <- read.table("putr_gmtc.csv", header=TRUE, sep=",")
putr_dups <- dup_checker(putr_data)

if(is.null(putr_dups)!=TRUE)
{
  setwd(out.dir)
  write.table(putr_dups, "duplicates_gmtc.csv", sep=",",
             row.names=FALSE)
}

putr_data <- read.table("putr_gmtc.csv", header=TRUE, sep=",")
putr_dups <- read.table("duplicates_gmtc.csv", header=TRUE, sep=",")

```

```

putr_data <- dup_replace(putr_data, putr_dups)
putr_data <- order_sample(putr_data)

setwd("output")
write.table(putr_data,
            paste("putr", "_gmtc.csv", sep=""),
            sep=" ", row.names=FALSE)

# #####
#
# Allele Characterization. #
#
# #####

# Initialize locus files.
putr_data <- read.table("putr_gmtc.csv", header=TRUE, sep=",")
gap <- 0.3
bin_init(putr_data, gap)

putr_data <- read.table("putr_gmtc.csv", header=TRUE, sep=",")
loci <- names(putr_data)[seq(3, ncol(putr_data), by=2)]
loci <- substr(loci, 1, nchar(loci)-1)
#
locus.l <- list()
for (i in 1:length(loci))
{
  locus.l[[i]]<-read.table(paste(loci[i], '_alleles.csv', sep=""),
                          header=TRUE, sep=',')
}
allele_process(putr_data, locus.l)

# #####
#
# Scoring Bins. #
#
# #####

putr_data <- read.table("putr_gmtc.csv", header=TRUE, sep=",")
loci <- names(putr_data)[seq(3,ncol(putr_data),by=2)]
loci <- substr(loci, 1, nchar(loci)-1)
#
bin_score(putr_data, loci,
          paste("putr", "_scored_gmtc.csv", sep=""))

# #####
#
# Export. #

```

```
#  
# #####  
  
# GenALEx Format  
putr_data <- read.table("putr_scored_gmtc.csv", header=TRUE, sep=",")  
putr_data <- genobins(putr_data)  
genomatic2genalex(putr_data, "putr")  
  
# NTSYSpc Format  
putr_data <- read.table("putr_scored_gmtc.csv", header=TRUE, sep=",")  
putr_data <- genobins(putr_data)  
genomatic2ntsys(putr_data, "putr")  
  
# Matrix Format  
putr_data <- read.table("putr_scored_gmtc.csv", header=TRUE, sep=",")  
putr_data <- genobins(putr_data)  
write.table(putr_data,  
            paste("putr", "_matrix.csv", sep=""),  
            sep=",", row.names=FALSE)
```

Appendix F

Troubleshooting the genomatic

My file doesn't load properly.

The genomatic requires several input file formats. Make sure to check file formats with the examples in appendix B. All rows in the files should be the same length. If there is any doubt about this you can always add a column of dummy numbers (e.g., 'NA's) to make sure they're all the same length. Extra columns beyond what is expected are usually ignored by the genomatic. Also make sure to use the 'check file' button to get an idea of how R and the genomatic will 'see' the file.

Requesting further help.

If you still can't get the genomatic to work, send me an email with the subject 'Genomatic: description of problem' to bknaus at fs dot fed dot us. Ideally, send me some files which reproduce the problem.