How_to_use_EFGLmh

Thomas Delomas

2022-01-28

Introduction

This will walk through the functions in the `EFGLmh` package. This package was written as a replacement for IDFGEN mainly motivated by the need to work with microhaps. It has been written to function for any codominant diploid marker, but is written and tested with SNPs and microhaps (SNPs are really just a subcategory of microhaps) in mind.

If you want to see all the options for a function, the manual has this information in a quicker-to-find format than this vignette.

One of the main differences between IDFGEN and `EFGLmh` is that IDFGEN keeps data in a separate environment, as if the data is somewhere in the ether until using an IDFGEN function to access it. This (mostly) prevents users from accidentally modifying things, but it also causes some bugginess, makes it hard for users to purposefully modify objects, and can cause some issues when loading in data from multiple data files. `EFGLmh` instead holds data as objects of a new class, called `EFGLdata`. As such, your dataset will have a variable name associated with it.

Now, let's walk through the functions in the package.

Getting data into EFGLmh

First, install if needed, and load the package.

```
# install tidyverse if you haven't already
install.packages("tidyverse")

# install the package if needed
devtools::install_github("delomast/EFGLmh")

# and load the package
library(EFGLmh)
# and load the tidyverse for the examples here
library(tidyverse)
options(tibble.max_extra_cols = 10) # one of my preferred options for tidyverse
```

Loading in data

Now, let's first load in our data. We can either read our data into as a dataframe, matrix, or tibble, perhaps do some modifications, and then hand it over to `EFGLmh`, or we can load it directly from a tab separated file into `EFGLmh`.

If we read it in separately, the dataframe, matrix, or tibble (below, variable `exampleData`) must have a column of population (pedigree) names, a column of unique individual names, optional metadata columns, and then genotype columns in a two column per call format. The pedigree and individual names can be anywhere, if their locations are specified. The genotype columns must be consecutive and on the right hand side. The start of the genotype columns can be specified, but if not, it will look for the first column with a column name ending in ".A1", ".a1", "-A1", or "-a1". Locus names are pulled from the first column for each locus.

```
# example of loading from file outside of EFGLmh
exampleData <- readr::read_tsv("example_snp_mh.txt", guess_max = 1e4)
```

```
print(exampleData)
#> # A tibble: 4,881 x 744
#>    Pedigree `Individual Nam~ Gender DateSampled LengthFork1 FieldID1 GenMa GenPa
#>    <chr>    <chr>            <chr>  <date>            <dbl> <chr>    <chr> <chr>
#>  1 OmyOXBO~ OmyOXBO19S_0001  F      2019-03-11          710 0001     <NA>  <NA>
#>  2 OmyOXBO~ OmyOXBO19S_0002  F      2019-03-11          680 0002     OmyO~ OmyO~
#>  3 OmyOXBO~ OmyOXBO19S_0003  F      2019-03-11          690 0003     OmyO~ OmyO~
#>  4 OmyOXBO~ OmyOXBO19S_0004  F      2019-03-11          700 0004     OmyO~ OmyO~
#>  5 OmyOXBO~ OmyOXBO19S_0005  F      2019-03-11          700 0005     <NA>  <NA>
#>  6 OmyOXBO~ OmyOXBO19S_0006  F      2019-03-11          680 0006     OmyO~ OmyO~
#>  7 OmyOXBO~ OmyOXBO19S_0007  F      2019-03-11          650 0007     OmyO~ OmyO~
#>  8 OmyOXBO~ OmyOXBO19S_0008  F      2019-03-11          730 0008     OmyO~ OmyO~
#>  9 OmyOXBO~ OmyOXBO19S_0009  F      2019-03-11          660 0009     <NA>  <NA>
#> 10 OmyOXBO~ OmyOXBO19S_0010  F      2019-03-14          740 0010     OmyO~ OmyO~
#> # ... with 4,871 more rows, and 736 more variables: OMS00039_mh-A1 <chr>,
#> #   OMS00039_mh-A2 <chr>, OMS00052_mh-A1 <chr>, OMS00052_mh-A2 <chr>,
#> #   OMS00077_mh-A1 <chr>, OMS00077_mh-A2 <chr>, OMS00101_mh-A1 <chr>,
#> #   OMS00101_mh-A2 <chr>, OMS00116_mh-A1 <chr>, OMS00116_mh-A2 <chr>, ...
# example of modifying in r, here we are just selecting a few populations
t1 <- exampleData %>% filter(Pedigree %in% c("OmyOXBO19S", "OmyEFSW19S"))
# or, without dplyr:
# t1 <- t[t$Pedigree %in% c("OmyOXBO19S", "OmyEFSW19S"),]

# and now we pass the data to EFGLmh
data1 <- readInData(t1)
```

If we want to read data directly from a file, we just specify the file name as the first argument. The file must have the same structure as described above: a column of population (pedigree) names, a column of unique individual names, optional metadata columns, and then genotype columns in a two column per call format. The pedigree and individual names can be anywhere, if their locations are specified. The genotype columns must be consecutive and on the right hand side. The start of the genotype columns can be specified, but if not, it will look for the first column with a column name ending in ".A1", ".a1", "-A1", or "-a1". Locus names are pulled from the first column for each locus. This has been chosen to work directly with Progeny outputs.

```
data_direct_from_file <- readInData("example_snp_mh.txt")
# metadata columns withs lots of blanks can give parsing errors.
# this can usually be solved with a larger number for the guess_max argument:
# data_direct_from_file <- readInData("example_snp_mh.txt", guess_max = 1e5)
```

There are many optional arguments for `readInData`:

- `genotypeStart`: you can specify the first genotype column if you don't want it to be auto-detected.
- `pedigreeColumn`: The column number that contains population (pedigree) names. Default is `1`
- `nameColumn`: The column number that contains individual names. Default is `2`
- `convertNames`: `TRUE` to convert genotype and pedigree names in the same way that IDFGEN does (remove special characters from both and remove "." from genotype names). Default is `TRUE`
- `convertMetaDataNames`: `TRUE` to remove special characters and spaces from metadata column names. This makes accessing them easier. Default is `TRUE`
- `missingAlleles`: a vector of values to treat as missing alleles. Default is `c("0", "00", "000")`
- `guess_max`: when reading from a file, this is passed to `read_tsv`. Parsing errors due to column data types can sometimes be fixed by increasing this. Default is 1e4

The `readInData` function creates object of class `EFGLdata`. When we `print` them, we just see a list of the population names and the number of loci.

```
data1
#> Populations:
#> [1] "OmyEFSW19S" "OmyOXBO19S"
#>
#>  368 loci
```

Structure of EFGLdata objects

The underlying structure of an `EFGLdata` object is just a list of two tibbles. The first entry is named "genotypes" and contains… metadata! No, it contains population names, individual names, and genotypes in a two column per call format (missing genotype is `NA`). The second entry is named "metadata" and contains population names, individual names, and metadata.

```
names(data1)
#> [1] "genotypes" "metadata"
data1$genotypes
#> # A tibble: 501 x 738
#>    Pop        Ind    OMS00039_mh.A1 OMS00039_mh.A2 OMS00052_mh.A1 OMS00052_mh.A2
#>    <chr>      <chr>  <chr>          <chr>          <chr>          <chr>
#>  1 OmyEFSW19S OmyEF~ GGC            GGC            GG             TG
#>  2 OmyEFSW19S OmyEF~ GAC            GAC            GG             GA
#>  3 OmyEFSW19S OmyEF~ GGC            GGC            GG             TG
#>  4 OmyEFSW19S OmyEF~ GGC            GAC            GG             GA
#>  5 OmyEFSW19S OmyEF~ GGC            GAC            GG             GA
#>  6 OmyEFSW19S OmyEF~ GGC            GAC            GG             GG
#>  7 OmyEFSW19S OmyEF~ GGC            GAC            GG             TG
#>  8 OmyEFSW19S OmyEF~ GGC            GAC            GG             TG
#>  9 OmyEFSW19S OmyEF~ GGC            GAC            TG             TG
#> 10 OmyEFSW19S OmyEF~ GGC            GGC            GG             GG
#> # ... with 491 more rows, and 732 more variables: OMS00077_mh.A1 <chr>,
#> #   OMS00077_mh.A2 <chr>, OMS00101_mh.A1 <chr>, OMS00101_mh.A2 <chr>,
#> #   OMS00116_mh.A1 <chr>, OMS00116_mh.A2 <chr>, OMS00118_mh.A1 <chr>,
#> #   OMS00118_mh.A2 <chr>, OMS00120_mh.A1 <chr>, OMS00120_mh.A2 <chr>, ...
data1$metadata
```

```
#> # A tibble: 501 x 8
#>    Pop        Ind       Gender DateSampled LengthFork1 FieldID1 GenMa    GenPa
#>    <chr>      <chr>     <chr>  <date>            <dbl> <chr>    <chr>    <chr>
#>  1 OmyEFSW19S OmyEFSW~ F      2019-04-07          710 0001     <NA>     <NA>
#>  2 OmyEFSW19S OmyEFSW~ M      2019-04-07          570 0003     <NA>     <NA>
#>  3 OmyEFSW19S OmyEFSW~ M      2019-04-10          640 0004     <NA>     <NA>
#>  4 OmyEFSW19S OmyEFSW~ F      2019-04-12          680 0007     OmyEFSW~ OmyEFSW~
#>  5 OmyEFSW19S OmyEFSW~ M      2019-04-12          520 0011     OmyEFSW~ OmyEFSW~
#>  6 OmyEFSW19S OmyEFSW~ M      2019-04-15          610 0016     OmyEFSW~ OmyEFSW~
#>  7 OmyEFSW19S OmyEFSW~ F      2019-04-15          710 0017     OmyEFSW~ OmyEFSW~
#>  8 OmyEFSW19S OmyEFSW~ F      2019-04-15          570 0018     OmyEFSW~ OmyEFSW~
#>  9 OmyEFSW19S OmyEFSW~ M      2019-04-17          760 0020     OmyEFSW~ OmyEFSW~
#> 10 OmyEFSW19S OmyEFSW~ M      2019-04-20          580 0022     OmyEFSW~ OmyEFSW~
#> # ... with 491 more rows
```

This (hopefully) makes it simple for you to pull out or modify data manually if there is not a specially written function in the package addressing what you want to accomplish. Simply refer to `data1$genotypes` or `data1$metadata` and treat it as you would treat any dataframe or tibble. One thing to remember if you modify things manually: the genotypes and metadata tibbles in an `EFGLdata` object should have the same individuals in the same order. You can check that you haven't messed things up by using the `construct_EFGLdata` function (performs some basic checks).

```
# do some modification on data1
# ...
# and then run some error checking
data1 <- construct_EFGLdata(data1)
```

Summarizing data

There are some common data summaries you may want from your data that EFGLmh has special functions to address. We'll walk through them here.

Just accessing data

Get the names of all populations in an EFGLdata object

```
pop_names <- getPops(data1)
pop_names
#> [1] "OmyEFSW19S" "OmyOXBO19S"
```

Get the names of all individuals, or all individuals in a subset of populations

```
# all inds in data1
all_inds <- getInds(data1)
# just look at first 20
all_inds[1:20]
#>  [1] "OmyEFSW19S_0001" "OmyEFSW19S_0003" "OmyEFSW19S_0004" "OmyEFSW19S_0007"
```

```
#>  [5] "OmyEFSW19S_0011" "OmyEFSW19S_0016" "OmyEFSW19S_0017" "OmyEFSW19S_0018"
#>  [9] "OmyEFSW19S_0020" "OmyEFSW19S_0022" "OmyEFSW19S_0023" "OmyEFSW19S_0024"
#> [13] "OmyEFSW19S_0026" "OmyEFSW19S_0032" "OmyEFSW19S_0034" "OmyEFSW19S_0035"
#> [17] "OmyEFSW19S_0042" "OmyEFSW19S_0052" "OmyEFSW19S_0053" "OmyEFSW19S_0054"
# only inds in one pop
subset_inds <- getInds(data1, pops = c("OmyOXBO19S"))
# looking at first 20
subset_inds[1:20]
#>  [1] "OmyOXBO19S_0001" "OmyOXBO19S_0002" "OmyOXBO19S_0003" "OmyOXBO19S_0004"
#>  [5] "OmyOXBO19S_0005" "OmyOXBO19S_0006" "OmyOXBO19S_0007" "OmyOXBO19S_0008"
#>  [9] "OmyOXBO19S_0009" "OmyOXBO19S_0010" "OmyOXBO19S_0011" "OmyOXBO19S_0012"
#> [13] "OmyOXBO19S_0013" "OmyOXBO19S_0014" "OmyOXBO19S_0015" "OmyOXBO19S_0016"
#> [17] "OmyOXBO19S_0017" "OmyOXBO19S_0018" "OmyOXBO19S_0019" "OmyOXBO19S_0020"
```

Get the locus names

```
loci <- getLoci(data1)
# looking at first 20
loci[1:20]
#>  [1] "OMS00039_mh"       "OMS00052_mh"       "OMS00077_mh"       "OMS00101_mh"
#>  [5] "OMS00116_mh"       "OMS00118_mh"       "OMS00120_mh"       "OMS00128_mh"
#>  [9] "OMS00129_mh"       "OMS00143_mh"       "OMS00149_mh"       "OMS00151_mh"
#> [13] "OMS00175_mh"       "Omy_101832195_mh" "Omy_102867443_mh" "Omy_104519624_mh"
#> [17] "Omy_107336170_mh" "Omy_108007193_mh" "Omy_109243222_mh" "Omy_110201359_mh"
```

Get the metadata column names

```
meta_names <- getMeta(data1)
meta_names
#> [1] "Gender"      "DateSampled" "LengthFork1" "FieldID1"    "GenMa"
#> [6] "GenPa"
```

Get the number of individuals in all, or a subset of populations

```
countInds <- numInds(data1)
countInds
#> OmyEFSW19S OmyOXBO19S
#>         25        476
countInds_1 <- numInds(data1, pops = c("OmyOXBO19S"))
countInds_1
#> OmyOXBO19S
#>        476
```

dumpTable from IDFGEN is also included, as it is commonly used

```
dumpTable(geno_success, "genotyping_success.txt")
```

some common calculations

Calculate allelic richness. I recommend doing this if you load in a dataset of just SNPs to make sure every locus has ≤ 2 alleles, as expected.

```
allele_rich <- aRich(data1)
# this returns a tibble
allele_rich
#> # A tibble: 343 x 2
#>    locus          aRich
#>    <chr>          <int>
#>  1 M09AAC055          2
#>  2 M09AAD076          2
#>  3 M09AAE082          2
#>  4 M09AAJ163          2
#>  5 Ocl_gshpx357       1
#>  6 OMGH1PROM1SNP1     2
#>  7 OMS00002           2
#>  8 OMS00003           2
#>  9 OMS00006           2
#> 10 OMS00008           2
#> # ... with 333 more rows
# if you want to see counts of loci by allelic richness
allele_rich %>% count(aRich)
#> # A tibble: 5 x 2
#>   aRich     n
#>   <int> <int>
#> 1     1     8
#> 2     2   262
#> 3     3    54
#> 4     4    18
#> 5     5     1
# same thing, without tidyverse
table(allele_rich$aRich)
#>
#>   1   2   3   4   5
#>   8 262  54  18   1
```

Genotyping success of loci, as a proportion

```
loci_success <- lociSuccess(data1)
loci_success
#> # A tibble: 368 x 2
#>    locus          success
#>    <chr>            <dbl>
#>  1 M09AAC055        0.964
#>  2 M09AAD076        0.952
#>  3 M09AAE082        0.962
#>  4 M09AAJ163        0.904
#>  5 Ocl_gshpx357     0.962
#>  6 OMGH1PROM1SNP1   0.958
#>  7 OMS00002         0.902
```

```
#>  8 OMS00003         0.768
#>  9 OMS00006         0.964
#> 10 OMS00008         0.878
#> # ... with 358 more rows
```

Genotyping success of individuals (more about removing failed individuals later)

```
geno_success <- genoSuccess(data1)
# this returns a tibble with both success as a proportion, and as the number of missing genotypes
geno_success
#> # A tibble: 501 x 4
#>    Pop         Ind              success numFail
#>    <chr>       <chr>              <dbl>   <int>
#>  1 OmyEFSW19S OmyEFSW19S_0001   0.910      33
#>  2 OmyEFSW19S OmyEFSW19S_0003   0.880      44
#>  3 OmyEFSW19S OmyEFSW19S_0004   0.918      30
#>  4 OmyEFSW19S OmyEFSW19S_0007   0.916      31
#>  5 OmyEFSW19S OmyEFSW19S_0011   0.924      28
#>  6 OmyEFSW19S OmyEFSW19S_0016   0.932      25
#>  7 OmyEFSW19S OmyEFSW19S_0017   0.913      32
#>  8 OmyEFSW19S OmyEFSW19S_0018   0.916      31
#>  9 OmyEFSW19S OmyEFSW19S_0020   0.916      31
#> 10 OmyEFSW19S OmyEFSW19S_0022   0.913      32
#> # ... with 491 more rows
# or with just a subset of loci
subsetLoci <- getLoci(data1)
subsetLoci <- subsetLoci[!grepl("SEX", subsetLoci)]
geno_success_noSDY <- genoSuccess(data1, loci = subsetLoci)
geno_success_noSDY
#> # A tibble: 501 x 4
#>    Pop         Ind              success numFail
#>    <chr>       <chr>              <dbl>   <int>
#>  1 OmyEFSW19S OmyEFSW19S_0001   0.910      33
#>  2 OmyEFSW19S OmyEFSW19S_0003   0.880      44
#>  3 OmyEFSW19S OmyEFSW19S_0004   0.918      30
#>  4 OmyEFSW19S OmyEFSW19S_0007   0.916      31
#>  5 OmyEFSW19S OmyEFSW19S_0011   0.924      28
#>  6 OmyEFSW19S OmyEFSW19S_0016   0.932      25
#>  7 OmyEFSW19S OmyEFSW19S_0017   0.913      32
#>  8 OmyEFSW19S OmyEFSW19S_0018   0.916      31
#>  9 OmyEFSW19S OmyEFSW19S_0020   0.916      31
#> 10 OmyEFSW19S OmyEFSW19S_0022   0.913      32
#> # ... with 491 more rows
```

Calculate allele frequency. Note that only alleles present in a population are shown for that population. As such, if a locus is all missing genotypes in a population, it is omitted for that population.

```
allele_freq <- calcAF(data1)
allele_freq
#> # A tibble: 1,459 x 5
```

```
#>    Pop        locus          allele count  freq
#>    <chr>      <chr>          <chr>  <int> <dbl>
#>  1 OmyEFSW19S M09AAC055      C         46  0.92
#>  2 OmyEFSW19S M09AAC055      T          4  0.08
#>  3 OmyEFSW19S M09AAD076      C         19  0.38
#>  4 OmyEFSW19S M09AAD076      T         31  0.62
#>  5 OmyEFSW19S M09AAE082      G         43  0.86
#>  6 OmyEFSW19S M09AAE082      T          7  0.14
#>  7 OmyEFSW19S M09AAJ163      A         24  0.5
#>  8 OmyEFSW19S M09AAJ163      G         24  0.5
#>  9 OmyEFSW19S Ocl_gshpx357   G         50  1
#> 10 OmyEFSW19S OMGH1PROM1SNP1 A          8  0.16
#> # ... with 1,449 more rows
```

Calculate observed and expected heterozygosity within each population

```
heterozygosity <- calcHet(data1)
heterozygosity
#> # A tibble: 686 x 4
#>    Pop        locus          expHet obsHet
#>    <chr>      <chr>           <dbl>  <dbl>
#>  1 OmyEFSW19S M09AAC055       0.147  0.08
#>  2 OmyEFSW19S M09AAD076       0.471  0.52
#>  3 OmyEFSW19S M09AAE082       0.241  0.28
#>  4 OmyEFSW19S M09AAJ163       0.5    0.5
#>  5 OmyEFSW19S Ocl_gshpx357    0      0
#>  6 OmyEFSW19S OMGH1PROM1SNP1  0.269  0.24
#>  7 OmyEFSW19S OMS00002        0.493  0.48
#>  8 OmyEFSW19S OMS00003        0.340  0.348
#>  9 OmyEFSW19S OMS00006        0.493  0.32
#> 10 OmyEFSW19S OMS00008        0.147  0.16
#> # ... with 676 more rows
```

manipulating EFGLdata objects

These functions handle some common operations we perform on our datasets.

Combining EFGLdata objects. Let's say we have two data files to read in (e.g. one mixture and one baseline, or multiple PBT baselines). We then want to combine them for filtering, analysis, export, etc.

```
# creating a second input datafile
t2 <- exampleData %>% filter(!(Pedigree %in% c("OmyOXBO19S", "OmyEFSW19S")))

# and now creating a second EFGLdata object
data2 <- readInData(t2)

data2
#> Populations:
#> [1] "OmyDWOR19S" "OmyLSCR19S" "OmyLYON19S" "OmyPAHH19S" "OmySAWT19S"
#> [6] "OmyWALL19S"
```

```
#>
#>  368 loci
data1
#> Populations:
#> [1] "OmyEFSW19S" "OmyOXBO19S"
#>
#>  368 loci
```

So we have two EFGLdata objects, one with 6 populations and one with 2. Now, we combine them:

```
all_data <- combineEFGLdata(data1, data2, genoComb = "intersect", metaComb = "intersect")
all_data
#> Populations:
#> [1] "OmyDWOR19S" "OmyEFSW19S" "OmyLSCR19S" "OmyLYON19S" "OmyOXBO19S"
#> [6] "OmyPAHH19S" "OmySAWT19S" "OmyWALL19S"
#>
#>  368 loci
```

We've now created a third EFGLdata object with all 8 populations. The arguments genoComb and metaComb tell the function how to combine loci and metadata if they are different. Options are "intersect" to only keep the loci or metadata that are in both, or "union" to keep the loci or metadata that are in either (missing loci/metadata will be given values of `NA`). Note that you can combine an unlimited number of `EFGLdata` objects in one command. For example, for four objects: `all_data <- combineEFGLdata(data1, data2, data3, data4, genoComb = "intersect", metaComb = "intersect")`. Having many EFGLdata objects can use a lot of memory if you have large numbers of individuals/loci. So, if you are done with the original EFGLdata objects, you can remove them to free up some memory:

```
rm(data1)
rm(data2)
```

Moving all individuals in a subset of populations to a different (new or existing) population.

```
# say we want to combine OmyDWOR19S and OmyEFSW19S, and call it "newPop"
all_data <- movePops(all_data, pops = c("OmyDWOR19S", "OmyEFSW19S"), newName = "newPop")
all_data
#> Populations:
#> [1] "newPop"     "OmyLSCR19S" "OmyLYON19S" "OmyOXBO19S" "OmyPAHH19S"
#> [6] "OmySAWT19S" "OmyWALL19S"
#>
#>  368 loci
```

Moving a subset of individuals to a different (new or existing) population

```
numInds(all_data)
#>     newPop OmyLSCR19S OmyLYON19S OmyOXBO19S OmyPAHH19S OmySAWT19S OmyWALL19S
#>       1395        126         87        476       1278        864        655
toMove <- c("OmyWALL19S_0696", "OmyWALL19S_0697", "OmyWALL19S_0698")
all_data <- moveInds(all_data, inds = toMove, newName = "specialInds")
numInds(all_data)
```

```
#>      newPop  OmyLSCR19S  OmyLYON19S  OmyOXBO19S  OmyPAHH19S  OmySAWT19S
#>        1395         126          87         476        1278         864
#>  OmyWALL19S specialInds
#>         652           3
```

Remove loci

```
toRemove <- c("OMS00079", "Omy_Omyclmk43896")
all_data <- removeLoci(all_data, lociRemove = toRemove)
all_data
#> Populations:
#> [1] "newPop"      "OmyLSCR19S"  "OmyLYON19S"  "OmyOXBO19S"  "OmyPAHH19S"
#> [6] "OmySAWT19S"  "OmyWALL19S"  "specialInds"
#>
#>  366 loci
```

Remove individuals

```
numInds(all_data)
#>      newPop  OmyLSCR19S  OmyLYON19S  OmyOXBO19S  OmyPAHH19S  OmySAWT19S
#>        1395         126          87         476        1278         864
#>  OmyWALL19S specialInds
#>         652           3
toRemove <- c("OmyWALL19S_0001", "OmyWALL19S_0002")
all_data <- removeInds(all_data, inds = toRemove)
numInds(all_data)
#>      newPop  OmyLSCR19S  OmyLYON19S  OmyOXBO19S  OmyPAHH19S  OmySAWT19S
#>        1395         126          87         476        1278         864
#>  OmyWALL19S specialInds
#>         650           3
```

Remove populations

```
all_data
#> Populations:
#> [1] "newPop"      "OmyLSCR19S"  "OmyLYON19S"  "OmyOXBO19S"  "OmyPAHH19S"
#> [6] "OmySAWT19S"  "OmyWALL19S"  "specialInds"
#>
#>  366 loci
all_data <- removePops(all_data, pops = c("OmyLSCR19S", "OmySAWT19S"))
all_data
#> Populations:
#> [1] "newPop"      "OmyLYON19S"  "OmyOXBO19S"  "OmyPAHH19S"  "OmyWALL19S"
#> [6] "specialInds"
#>
#>  366 loci
```

exporting data

These functions export data in formats used by other packages and programs. They are listed here mainly so you can have a list of the export functions in one place and see an example. For a full explanation of the options within each of these functions, consult the manual. Most have options to subset loci and populations.

A rubias baseline

```
gsi_baseline <- exportRubias_baseline(all_data, pops = c("OmyOXBO19S", "newPop"),
                                      repunit = "Pop", collection = "Pop", loci = NULL)
```

A rubias mixture

```
gsi_mixture <- exportRubias_mixture(all_data, pops = c("OmyLYON19S", "specialInds"),
                                    collection = "Pop", loci = NULL)
```

A gRandma baseline or mixture

```
# baseline
gma_baseline <- exportGrandma(all_data, pops = c("OmyOXBO19S", "newPop"), loci = NULL,
                              baseline = TRUE)
# mixture
gma_mixture <- exportGrandma(all_data, pops = c("OmyLYON19S", "specialInds"), loci = NULL,
                              baseline = FALSE)
```

In addition to exporting gRandma inputs, there is also a function to remove loci for gRandma inputs that either failed for all individuals, or have no variation.

```
# this creates a list with baseline and mixture
cleanInput <- cleanGrandma(baseline = gma_baseline, mixture = gma_mixture)
#> Removing locus Ocl_gshpx357 for no variation.
#> Removing locus Omy_myclarp404111 for no variation.
#> Removing locus Omy_BAMBI4238 for no variation.
#> Removing locus Omy_G3PD_2246 for no variation.
#> Removing locus Omy_RAD10335945 for no variation.
gma_baseline <- cleanInput$baseline
gma_mixture <- cleanInput$mixture
```

A hierfstat input dataframe

```
hfstat_in <- exportHierFstat(all_data)
```

A CKMRsim allele frequency tibble. Note that loci with all missing genotypes will be removed. And all pops (or all pops specified with the `pops` argument) are combined.

```
ckmr_af <- exportCKMRsimAF(all_data)
```

A long format tibble of genotypes. Meant for input into CKMRsim. Note that pop information is not included in the export.

```
ckmr_af <- exportCKMRsimLG(all_data, pops = c("OmyOXBO19S"))
```

Write a GenePop file

```
exportGenePop(all_data, "genepop.txt", useIndNames = TRUE)
```

Write a GenAlEx file

```
exportGenAlEx(all_data, "genalexInput.txt")
```

Write a Structure file

```
exportStructure(all_data, "structureInput.txt")
```

Write a “Progeny-export-style” file that can be later loaded back into EFGLmh with `readInData()`. This file will have Pop, Ind, metadata, genotypes (2 column per call).

```
exportProgenyStyle(all_data, "progenyStyleFile.txt")
```

Write a SNPPIT input file (only biallelic markers used)

```
exportSNPPIT(all_data, "snppitInput.txt", baseline = c("OmyOXBO19S", "newPop"),
             mixture = c("OmyLYON19S", "specialInds"), errorRate = .005)
```

Write PLINK input files (only biallelic markers used)

```
exportPlink(all_data, "testPlink.ped", map = "testPlink.map")
```

examples of common steps in data analysis

remove poorly genotyping individuals

First, we determine genotyping success. We’re using all loci, but remember `genoSuccess` can also use just a subset of loci if you input a vector of locus names.

```
geno_success <- genoSuccess(all_data)
geno_success
#> # A tibble: 3,889 x 4
#>    Pop    Ind             success numFail
#>    <chr>  <chr>             <dbl>   <int>
#>  1 newPop OmyDWOR19S_0001   0.962      14
#>  2 newPop OmyDWOR19S_0002   0.995       2
```

```
#>  3 newPop OmyDWOR19S_0003   0.978       8
#>  4 newPop OmyDWOR19S_0004   0.995       2
#>  5 newPop OmyDWOR19S_0005   0.989       4
#>  6 newPop OmyDWOR19S_0006   0.992       3
#>  7 newPop OmyDWOR19S_0007   0.992       3
#>  8 newPop OmyDWOR19S_0008   0.989       4
#>  9 newPop OmyDWOR19S_0009   0.997       1
#> 10 newPop OmyDWOR19S_0010   0.989       4
#> # ... with 3,879 more rows
```

Now we get a list of individual names to remove and then use the `removeInds` function. We can filter by the proportion success or by the number of missing loci.

```
# identify any under 90%
failedInds <- geno_success %>% filter(success < .9) %>% pull(Ind)
# example of code to filter by number of missing loci
#  (remove any with more than 37 genotypes missing)
# failedInds <- geno_success %>% filter(numFail < 37) %>% pull(Ind)
# and remove
sum(numInds(all_data))
#> [1] 3889
all_data <- removeInds(all_data, inds = failedInds)
sum(numInds(all_data))
#> [1] 3448
```

And perhaps we want to save a list of the failed individuals and their populations

```
removeTable <- geno_success %>% filter(Ind %in% failedInds)
dumpTable(removeTable, "failed_inds.txt")
# or, to efficiently do it all in the tidyverse:
# geno_success %>% filter(Ind %in% failedInds) %>% dumpTable("failed_inds.txt")
```

removing duplicates (with some outside help)

Just as we use GSI_sim to quickly identify duplicate samples, we can use rubias.

```
rubiasIn <- exportRubias_baseline(all_data, pops = c("OmyPAHH19S", "specialInds"),
                                  repunit = "Pop", collection = "Pop")
# require 70% genotypes successful in both (but we've already filtered), and
# requie 95% of genotypes to be the same
library(rubias)
#> Warning: package 'rubias' was built under R version 4.1.2
dupTable <- close_matching_samples(rubiasIn, gen_start_col = 5, min_frac_non_miss = .7,
         min_frac_matching = .95)
#> Summary Statistics:
#>
#> 1009 Individuals in Sample
#>
#> 366 Loci: M09AAC055.A1, M09AAD076.A1, M09AAE082.A1, M09AAJ163.A1, Ocl_gshpx357.A1,
         OMGH1PROM1SNP1.A1, OMS00002.A1, OMS00003.A1, OMS00006.A1, OMS00008.A1, OMS00013.A1,
```

OMGHIPROM1SNP1.A1, OMS00002.A1, OMS00003.A1, OMS00006.A1, OMS00008.A1, OMS00013.A1,
OMS00014.A1, OMS00015.A1, OMS00017.A1, OMS00018.A1, OMS00024.A1, OMS00030.A1,
OMS00039_mh.A1, OMS00041.A1, OMS00048.A1, OMS00052_mh.A1, OMS00053.A1, OMS00056.A1,
OMS00057.A1, OMS00058.A1, OMS00061.A1, OMS00062.A1, OMS00064.A1, OMS00068.A1, OMS00070.A1,
OMS00071.A1, OMS00072.A1, OMS00074.A1, OMS00077_mh.A1, OMS00078.A1, OMS00087.A1,
OMS00089.A1, OMS00090.A1, OMS00092.A1, OMS00095.A1, OMS00096.A1, OMS00101_mh.A1,
OMS00103.A1, OMS00105.A1, OMS00106.A1, OMS00111.A1, OMS00112.A1, OMS00114.A1,
OMS00116_mh.A1, OMS00118_mh.A1, OMS00119.A1, OMS00120_mh.A1, OMS00121.A1, OMS00127.A1,
OMS00128_mh.A1, OMS00129_mh.A1, OMS00132.A1, OMS00133.A1, OMS00134.A1, OMS00138.A1,
OMS00143_mh.A1, OMS00149_mh.A1, OMS00151_mh.A1, OMS00153.A1, OMS00154.A1, OMS00156.A1,
OMS00164.A1, OMS00169.A1, OMS00173.A1, OMS00174.A1, OMS00175_mh.A1, OMS00176.A1,
OMS00179.A1, OMS00180.A1, Omy_1004.A1, Omy_101554306.A1, Omy_101832195_mh.A1,
Omy_101993189.A1, Omy_102505102.A1, Omy_102867443_mh.A1, Omy_103705558.A1,
Omy_104519624_mh.A1, Omy_104569114.A1, Omy_105075162.A1, Omy_105105448.A1, Omy_105385406.A1,
Omy_105714265.A1, Omy_107031704.A1, Omy_10728569.A1, Omy_107336170_mh.A1, Omy_10780634.A1,
Omy_108007193_mh.A1, Omy_109243222_mh.A1, Omy_109525403.A1, Omy_109894185.A1,
Omy_110064419.A1, Omy_110201359_mh.A1, Omy_110362585.A1, Omy_110689148.A1, Omy_111084526.A1,
Omy_11138351_mh.A1, Omy_111666301.A1, Omy_112301202_mh.A1, Omy_11282082_mh.A1,
Omy_113490159.A1, Omy_114315438.A1, Omy_114587480.A1, Omy_114976223.A1, Omy_116733349.A1,
Omy_116938264.A1, Omy_117286374.A1, Omy_117370400.A1, Omy_117540259_mh.A1, Omy_11781581.A1,
Omy_118175396.A1, Omy_118205116.A1, Omy_11865491.A1, Omy_120255332_mh.A1, Omy_128693455.A1,
Omy_128923433_mh.A1, Omy_128996481.A1, Omy_129870756.A1, Omy_130524160_mh.A1,
Omy_131460646_mh.A1, Omy_187760385_mh.A1, Omy_96222125_mh.A1, Omy_9707773_mh.A1,
Omy_97660230.A1, Omy_97865196.A1, Omy_97954618_mh.A1, Omy_98683165.A1, Omy_99300202_mh.A1,
Omy_ada1071.A1, Omy_anp17_mh.A1, Omy_aromat280_mh.A1, Omy_arp630.A1, Omy_aspAT123_mh.A1,
Omy_b1266.A1, Omy_b9164.A1, Omy_BACB4324.A1, Omy_BACF5284_mh.A1, Omy_BAMBI2312.A1,
Omy_BAMBI4238.A1, Omy_bcAKala380rd.A1, Omy_ca05064_mh.A1, Omy_carban1264_mh.A1,
Omy_cd28130.A1, Omy_cd59206.A1, Omy_cd59b112.A1, Omy_cin172.A1, Omy_cox1221.A1,
Omy_cox2335_mh.A1, Omy_crb106.A1, Omy_cyp17153_mh.A1, Omy_e1147_mh.A1, Omy_ftzf1217.A1,
Omy_g1103.A1, Omy_g1282_mh.A1, Omy_G3PD_2246.A1, Omy_G3PD_2371.A1, Omy_gadd45332.A1,
Omy_gdh271.A1, Omy_GH1P1_2.A1, Omy_gh475.A1, Omy_GHSR121_mh.A1, Omy_gluR79.A1,
Omy_GREB1_05.A1, Omy_GREB1_09.A1, Omy_hsc71580_mh.A1, Omy_hsf1b241.A1, Omy_hsf2146.A1,
Omy_hsp4786.A1, Omy_hsp70aPro329.A1, Omy_hsp90BA193.A1, Omy_hus152.A1, Omy_IL17185.A1,
Omy_Il1b_028.A1, Omy_IL1b163_mh.A1, Omy_IL6320.A1, Omy_impa155.A1, Omy_inos97.A1,
Omy_LDHB1_i2.A1, Omy_LDHB2_e5.A1, Omy_LDHB2_i6.A1, Omy_lpl220_mh.A1, Omy_mapK3103.A1,
Omy_mcsf268_mh.A1, Omy_metA161_mh.A1, Omy_metB138_mh.A1, Omy_MYC_2_mh.A1,
Omy_myclarp404111.A1, Omy_myoD178.A1, Omy_nach200_mh.A1, Omy_NaKATPa350.A1, Omy_ndk152.A1,
Omy_nips299.A1, Omy_nkef241_mh.A1, Omy_ntl27.A1, Omy_nxt2273_mh.A1, Omy_Ogo4212.A1,
Omy_OmyP9180.A1, Omy_Ots249227.A1, Omy_oxct85_mh.A1, Omy_p53262.A1, Omy_pad196_mh.A1,
Omy_ppie232_mh.A1, Omy_RAD10335945.A1, Omy_RAD1073310.A1, Omy_RAD11659.A1, Omy_RAD118659.A1,
Omy_RAD1243964.A1, Omy_RAD1256614.A1, Omy_RAD1303467.A1, Omy_RAD1307316.A1,
Omy_RAD1349913.A1, Omy_RAD1403346.A1, Omy_RAD1570953.A1, Omy_RAD1610420_mh.A1,
Omy_RAD1763223.A1, Omy_RAD1784916.A1, Omy_RAD1890348_mh.A1, Omy_RAD191922.A1,
Omy_RAD1934024.A1, Omy_RAD1957859.A1, Omy_RAD2091711.A1, Omy_RAD2212369.A1,
Omy_RAD2357743.A1, Omy_RAD2389458_mh.A1, Omy_RAD2428774.A1, Omy_RAD2504268.A1,
Omy_RAD25678_mh.A1, Omy_RAD2608069_mh.A1, Omy_RAD2669136_mh.A1, Omy_RAD2774055.A1,
Omy_RAD2823638.A1, Omy_RAD2970018_mh.A1, Omy_RAD297626_mh.A1, Omy_RAD3039217.A1,
Omy_RAD3061961.A1, Omy_RAD3140867.A1, Omy_RAD320910.A1, Omy_RAD3213958.A1,
Omy_RAD3312247.A1, Omy_RAD3379824.A1, Omy_RAD3500513.A1, Omy_RAD351499.A1, Omy_RAD354179.A1,
Omy_RAD365148.A1, Omy_RAD3667.A1, Omy_RAD368487.A1, Omy_RAD3695253.A1, Omy_RAD3781668.A1,
Omy_RAD3840619.A1, Omy_RAD3915633.A1, Omy_RAD392622_mh.A1, Omy_RAD4013255.A1,
Omy_RAD4052048.A1, Omy_RAD4064158.A1, Omy_RAD4159434.A1, Omy_RAD4246532.A1,
Omy_RAD4279359.A1, Omy_RAD4357337.A1, Omy_RAD4361242_mh.A1, Omy_RAD4369441.A1,
Omy_RAD4510418.A1, Omy_RAD4631435.A1, Omy_RAD4645251.A1, Omy_RAD4667227.A1,
Omy_RAD4708054.A1, Omy_RAD4744453_mh.A1, Omy_RAD4795551.A1, Omy_RAD484814.A1,
Omy_RAD4879969.A1, Omy_RAD4911135_mh.A1, Omy_RAD5063221.A1, Omy_RAD5245817_mh.A1,
Omy_RAD5281228_mh.A1, Omy_RAD537456_mh.A1, Omy_RAD5540454.A1, Omy_RAD5599710.A1,
Omy_RAD5791629.A1, Omy_RAD5821370_mh.A1, Omy_RAD5883515_mh.A1, Omy_RAD5975841.A1,
Omy_RAD5995044.A1, Omy_RAD6013512.A1, Omy_RAD61959.A1, Omy_RAD6259638.A1, Omy_RAD6580868.A1,
Omy_RAD6595969.A1, Omy_RAD6640236.A1, Omy_RAD6683417.A1, Omy_RAD6863440.A1,
Omy_RAD701631_mh.A1, Omy_RAD72108_mh.A1, Omy_RAD7252844_mh.A1, Omy_RAD7320463_mh.A1,
Omy_RAD738450_mh.A1, Omy_RAD73959.A1, Omy_RAD7396373_mh.A1, Omy_RAD7606020.A1,
Omy_RAD7657062_mh.A1, Omy_RAD7778954.A1, Omy_RAD7814727.A1, Omy_RAD7850257.A1,
Omy_RAD7877610.A1, Omy_RAD7931458_mh.A1, Omy_RAD8513135.A1, Omy_RAD8670672_mh.A1,
Omy_RAD880287.A1, Omy_RAD8812232.A1, Omy_RAD900413_mh.A1, Omy_RAD9248564_mh.A1,
Omy_RAD9358037.A1, Omy_RAD9871553.A1, Omy_rapd167.A1, Omy_rbm4b203.A1, Omy_redd1410.A1,
Omy_sast264_mh.A1, Omy_SECC22b88_mh.A1, Omy_srp0937.A1, Omy_sSOD1.A1, Omy_star206.A1,

```
         Omy_sast264_mh.A1, Omy_SECC22b88_mh.A1, Omy_srp0937.A1, Omy_sSOD1.A1, Omy_star206.A1,
         Omy_stat3273.A1, Omy_sys1188_mh.A1, Omy_tlr3377.A1, Omy_tlr5205_mh.A1, Omy_txnip343_mh.A1,
         Omy_u0779166.A1, Omy_u0953469.A1, Omy_u0954311.A1, Omy_u0956119_mh.A1, Omy_u0961043.A1,
         Omy_U11_2b154_mh.A1, Omy_UBA3b.A1, Omy_UT16_2173.A1, Omy_vamp5303.A1, Omy_vatf406.A1,
         Omy_zg5791.A1, OMY1011SNP.A1, Omy25_61284413.A1, Omy25_61285646.A1, Omy25_61286316.A1,
         Omy25_61287415.A1, Omy25_61294400.A1, Omy25_61316270.A1, Omy25_61317685.A1,
         Omy25_61317777.A1, Omy25_61318852.A1, Omy25_61322413.A1, Omy28_11607954.A1,
         Omy28_11625241.A1, Omy28_11632591.A1, Omy28_11658853.A1, Omy28_11667578.A1,
         Omy28_11671116.A1, Omy28_11676622.A1, Omy28_11683204.A1, Omy28_11773194.A1, OmyR14589.A1,
         OmyR19198.A1, OmyR24370.A1, OmyR33562.A1, OmyR40252.A1, OmyR40319.A1, OmyY1_2SEXY.A1
#>
#> 2 Reporting Units: OmyPAHH19S, specialInds
#>
#> 2 Collections: OmyPAHH19S, specialInds
#>
#> 8.85% of allelic data identified as missing
dupTable
#> # A tibble: 2 x 10
#>   num_non_miss num_match indiv_1 indiv_2 collection_1 collection_2 sample_type_1
#>          <int>     <int> <chr>   <chr>   <chr>        <chr>        <chr>
#> 1          335       335 OmyPAH~ OmyPAH~ OmyPAHH19S   OmyPAHH19S   reference
#> 2          336       336 OmyPAH~ OmyPAH~ OmyPAHH19S   OmyPAHH19S   reference
#> # ... with 3 more variables: repunit_1 <chr>, sample_type_2 <chr>,
#> #   repunit_2 <chr>
```

So we’ve found two pairs of duplicates, now we want to keep the ones with more genotypes. We can use the genotyping success calculated earlier to choose which one to remove. There is a special function to identify the one with lower genotyping success.

```
toRemove <- whichLower(dupTable, geno_success)
all_data <- removeInds(all_data, inds = toRemove)
```

And let’s write a table of the duplicates

```
dupTable %>% select(1:6) %>% dumpTable("duplicates.txt")
```

simulating data

A basic method of simulating F1 hybrids resulting from mating between two populations is available. Samples alleles based on allele frequencies in the two populations. If your data has a sex marker in it, you probably want to remove it prior to using this function.

```
# remove sex marker, then simulate 125 F1 hybrids
data1_with_hyb <- all_data %>% removeLoci(lociRemove = "OmyY1_2SEXY") %>%
  createF1Hybrids(pop1 = "OmyWALL19S", pop2 = "OmyOXBO19S", newName = "fakeHybridsF1",
                  n = 125, missingGenos = TRUE)
data1_with_hyb
#> Populations:
#> [1] "fakeHybridsF1" "newPop"        "OmyLYON19S"    "OmyOXBO19S"
#> [5] "OmyPAHH19S"    "OmyWALL19S"    "specialInds"
```

```
#>
#>  365 loci
```