Getting Genetic Data Into R

Rodney J. Dyer
Department of Biology
Virginia Commonwealth University
http://dyerlab.bio.vcu.edu

Synopsis

Here you will learn to get genetic data files into the R environment using the `gstudio` package. This package was designed to handle marker-based genetic data (e.g., not sequences *per se* though it can use SNP's and haplotypes) as well as additional data that is typically collected along with individuals.

To get started, first import the `gstudio` package as:

```
> require(gstudio)
```

The `Locus` Class

The locus class is the fundamental class that handles marker-based genetic data. At present it can handle dominant and co-dominant marker types at any ploidy level. Internally, alleles are stored as a `character` vector and by default they are not sorted so that the alleles will be presented in the order that you import them (e.g., a `3:1` locus instead of a `1:3` locus). I do not sort these because it may be necessary to know the phase of the alleles in a locus and sorting them would remove that information. If you abhor the sight of a genotype `3:1` then sort it earlier and then try to figure out why you have this affliction.

```
> loc1 <- Locus( c(120,122) )
> loc1
```

```
120:122
```

```
> loc2 <- Locus( c("A","T") )
> loc2
```

```
A:T
```

Note, that internally the alleles are translated into `character` objects. In all the functions dealing with alleles both `integer` and `character` arguments are accepted. There are several methods associated with the `Locus`, the main ones that you will be working with are shown below by example. See `help("Locus-class")` for a complete discussion.

```
> loc3 <- Locus( c(122,122) )
> loc3
```

```
122:122
```

```
> is.heterozygote( loc3 )
```

```
[1] FALSE
```

```
> loc3[2]
```

```
[1] "122"

> loc3[2] <- "124"
> is.heterozygote( loc3 )

[1] TRUE

> length( loc3 )

[1] 2

> summary( loc3 )

Class : Locus
Ploidy : 2
Aleleles : 122,124
```

Another useful method of the `Locus` class is the `as.multivariate` function. This translates the locus into a multivariate coding vector so you can do some real statistics with it. Here is an example:

```
> loc4 <- Locus( c("A","C") )
> loc4

A:C

> all.alleles <- c("A","G","C","T")
> all.alleles

[1] "A" "G" "C" "T"

> as.vector( loc4, all.alleles )

[1] 1 0 1 0
```

The `Population` Class

You can think of a `Population` is a collection of one or more individuals. While no man is an island, an individual is just a population of $N = 1$. Each individual, can have any number of `Locus` objects along with other non-genetic information associated with them (e.g., latitude, longitude, dbh, hair color, etc.). You create a population by passing it data columns in much the same way as how you create a `data.frame` (in fact, the `Population` class is just a `data.frame` that knows how to deal with `Locus` objects and how to give you population genetic summaries).

```
> strata <- c("A","A","B","B","B")
> TPI <- c(Locus(c(1,2)),Locus(c(2,3)),Locus(c(2,2)),Locus(c(2,2)),Locus(c(1,3)))
> PGM <- c(Locus(c(4,4)),Locus(c(4,3)),Locus(c(4,4)),Locus(c(3,4)),Locus(c(3,3)))
> Env <- c(12,20,14,18,10)
> thePop <- Population( Pop=strata, Env=Env, TPI=TPI, PGM=PGM )
> thePop

  Pop Env TPI PGM
1   A  12 1:2 4:4
2   A  20 2:3 3:4
3   B  14 2:2 4:4
4   B  18 2:2 3:4
5   B  10 1:3 3:3

> summary(thePop)
```

```
     Pop                  Env          TPI      PGM
 Length:5           Min.   :10.0   1:2:1   3:3:1
 Class :character   1st Qu.:12.0   1:3:1   3:4:2
 Mode  :character   Median :14.0   2:2:2   4:4:2
                    Mean   :14.8   2:3:1
                    3rd Qu.:18.0
                    Max.   :20.0
```

```
> names(thePop)
```

```
[1] "Pop" "Env" "TPI" "PGM"
```

Accessing `Population` Elements

You can also add data to a `Population` or remove it

```
> WXY <- c(Locus(c(122,124)),Locus(c(124,126)),Locus(c(124,124)),Locus(c(122,124)),Locus(c(126,126)))
> thePop$WXY <- WXY
> thePop
```

```
  Pop Env TPI PGM     WXY
1   A  12 1:2 4:4 122:124
2   A  20 2:3 3:4 124:126
3   B  14 2:2 4:4 124:124
4   B  18 2:2 3:4 122:124
5   B  10 1:3 3:3 126:126
```

```
> thePop$WXY <- NULL
> thePop
```

```
  Pop Env TPI PGM
1   A  12 1:2 4:4
2   A  20 2:3 3:4
3   B  14 2:2 4:4
4   B  18 2:2 3:4
5   B  10 1:3 3:3
```

Similar to the previous constructs, you can access elements within a `Population` using either numerical indexes, slices, or names.

```
> ind3 <- thePop[3,]
> ind3
```

```
  Pop Env TPI PGM
1   B  14 2:2 4:4
```

```
> thePop[ thePop$Pop=="B", ]
```

```
  Pop Env TPI PGM
1   B  14 2:2 4:4
2   B  18 2:2 3:4
3   B  10 1:3 3:3
```

```
> thePop[ thePop$Env<15 , ]
```

```
  Pop Env TPI PGM
1   A  12 1:2 4:4
2   B  14 2:2 4:4
3   B  10 1:3 3:3
```

```
> TPI <- thePop[,3]
> print(TPI)

[[1]]
1:2

[[2]]
2:3

[[3]]
2:2

[[4]]
2:2

[[5]]
1:3
```

Getting Data Types within `Population` Objects

Since a `Population` can hold several types of data and the main way to get data from one is to know its name, the method `column.names` can provide you quick access to all the data names of a specific R `class`.

```
> strata <- column.names(thePop,"character")
> strata

[1] "Pop"

> column.names(thePop,"Locus")

[1] "TPI" "PGM"

> column.names(thePop,"numeric")

[1] "Env"
```

Partitioning `Population` Objects

A `Population` object can contain individuals with several other categorical data variables (e.g., population, region, habitat, etc.) and it is relatively easy to get single elements (as shown in the slicing above) as well as complete partitions. It should be pointed out that when you `partition` a `Population` on some stratum, it will remove that stratum from all the partitions though it will leave the other partitions in the subpopulations.

```
> subpops <- partition(thePop,stratum="Pop")
> print(subpops)

$A
  Env TPI PGM
1  12 1:2 4:4
2  20 2:3 3:4

$B
  Env TPI PGM
1  14 2:2 4:4
2  18 2:2 3:4
3  10 1:3 3:3
```

Generic `Population` Functions

The following generic functions are available for the `Population` class and work just like they do using other data structures.

length The number of `Individual` objects (rows) in the `Population`.

dim The number or row and columns in the `Population`.

names The data column names.

summary A summary of the data columns in the `Population`.

show Dumps the `Population` to the terminal.

row.names Returns the names of the rows (they are integers so this isn't too exciting).

Importing Data

OK, so typing all this stuff in is rather monotonous and will be a total pain if you have a real data set with hundreds or thousands of individuals and a righteous amount of loci.

The main function for importing data from a text file into a `Population` object is `read.population` and assumes the following about your data:

1. You have your data in a TEXT file that is comma separated (*.csv).
2. You have a header row on your file with the names of each column of data. Headers should not have spaces in them, R will replace them with a period.
3. Genetic marker that have more than one allele are encoded using a colon ":" separating alleles. This means that the diploid microsatellite locus with alleles 122 & 128 would be in a single column as 122:128. This allows you to have triploid, tetraploid, etc markers with not other encoding.
4. Haploid markers are do not need a ":", just put in the haplotype. With haploid data, searching for ":" won't work so you need to pass the number of haploid loci as the optional parameter `num.haploid` to `read.population`. The haploid loci *must* be the last `num.haploid` right-most columns in your data set.
5. All alleles will be treated internally as a `character` string (except for in a few cases such as estimating ladder-distance). So you can use all alphanumeric characters for alleles but stay away from punctuation.
6. Missing data should be encoded as NA (for the whole genotype NA:NA is just silly).
7. If you have a mixture of genetic data types, columns with ":" will be automatically interpreted as `Locus` objects. You can mix in haploid data types by putting them in the last, right-most, columns and pass the optional parameter `num.haploid` with the number columns to put as haploid.

Reading From a Text File

An example data file may look like:

```
Population,Lat,Lon,PGM,TPI
Loreto,22.25,-102.01,120:122,A:T
Loreto,22.25,-102.01,122:124,A:C
Cabo,22.88,-109.9,120:120,A:A
Cabo,22.88,-109.9,NA,A:T
```

This file can be loaded as (assuming `getwd()` contains the file)

```
> pop <- read.population(file="testData.csv")
> summary(pop)
```

```
  Population      Lat              Lon               PGM      TPI
 Cabo   :2    Min.   :22.25   Min.   :-109.9   120:120:1   A:A:1
 Loreto:2     1st Qu.:22.25   1st Qu.:-109.9   120:122:1   A:C:1
              Median :22.57   Median :-106.0   122:124:1   A:T:2
              Mean   :22.57   Mean   :-106.0   NA's   :1
              3rd Qu.:22.88   3rd Qu.:-102.0
              Max.   :22.88   Max.   :-102.0
```

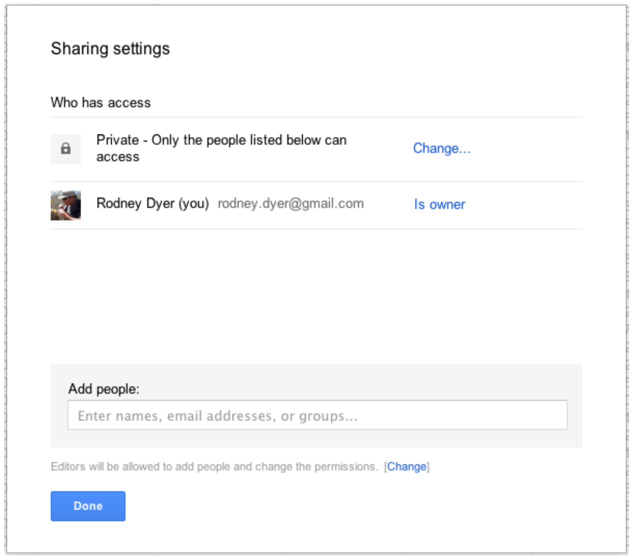
In general, if you can open your file using `read.table`, then `read.population` should work.

Using Google Spreadsheets To Share Data

One of the really great things about google docs is that you can use it to share information and documents with others and here we will be examining how to use it to keep public data available for analysis in R.

The first step is to provide a bit of data to share. The following example uses the shared *Cornus florida* data set. This consists of adults and offspring.

To share a document, click the ”Share” button and you will be presented with a popup window giving you options on what to do similar to Figure .



Sharing settings

Who has access

Private - Only the people listed below can access — Change...

Rodney Dyer (you) rodney.dyer@gmail.com — Is owner

Add people:

Enter names, email addresses, or groups...

Editors will be allowed to add people and change the permissions. [Change]

Done

Figure 1: Settings to adjust sharing options for google document.

Where it says Private select the ”Change...” option and change the Visibility Options to ”Anyone with the link” and hit save. It will then return to the Sharing Settings (Figure) page and provide you a unique link to the document.

This gives individuals access to the spreadsheet as a whole, but what we would like to do is to get to the contents of it as a *.csv file. In the spreadsheet, select `File` → `Publish to the Web` and select the following options in the dialog:

1. Sheets to Publish → `All sheets`
2. Check the box `Automatically republish when changes are made`
3. Select `Start publishing`.

This will make the bottom part of the dialog active and you'll need to make the following changes:

1. Change type from `Web` → `CSV`
2. Change `All Sheets` → `Sheet1`
3. Change `All Cells` → the range that you want to share. Here you need to use Excel-like notation such as `A1:I63` for the box from column A, first row to column I, 63nd row.

The dialog provides a URL for these data, the one above is:

```
https://docs.google.com/spreadsheet/pub?hl=en_US&hl=en_US&key=0Aq-lsUWPDuZtdF9xMXZGQWNtbk1F
NTVWd3F3U0FDdXc&single=true&gid=0&range=A1%3AG63&output=csv
```

1 Getting Data Into R from GoogleDocs

Now we have a data set that is available on the web and we can get to it from within R using the the `getURL`, `read.csv`, and `textConnection` functions as follows (n.b. I truncated the URL as it goes off the end of the page, it is the one from above.)

```
> spreadsheetURL <- "https://docs.google.com/spreadsheet/pub?hl=en_US&hl=en_US&key=0Aq-..."
> dogwood <- read.population( googleURL=dogwoodURL )
```

And there you go, you have now used your Google Account to host data that is available to everyone... No go forth and share.

Example Data Sets

The `gstudio` package comes with some example data sets already loaded. To access these data sets, use the `data` function and they will be put into your workspace (already formatted as `Population` objects).

```
> data(araptus_attenuatus)
> summary(araptus_attenuatus)
```

```
   Species      Cluster         Pop        Individual       Lat
 CladeA: 75   CBP-C :150   32      : 19   101_10A:  1   Min.   :23.08
 CladeB: 36   NBP-C : 84   75      : 11   101_1A :  1   1st Qu.:24.59
 CladeC:252   SBP-C : 18   Const   : 11   101_2A :  1   Median :26.25
              SCBP-A: 75   12      : 10   101_3A :  1   Mean   :26.25
              SON-B : 36   153     : 10   101_4A :  1   3rd Qu.:27.53
                           157     : 10   101_5A :  1   Max.   :29.33
                           (Other):292   (Other):357
      Long           LTRS          WNT           EN          EF
 Min.   :-114.3   01:01:147   03:03   :108   01:01   :225   01:01:219
 1st Qu.:-113.0   01:02: 86   01:01   : 82   01:02   : 52   01:02: 52
 Median :-111.5   02:02:130   01:03   : 77   02:02   : 38   02:02: 90
 Mean   :-111.7               02:02   : 62   03:03   : 22   NA   :  2
 3rd Qu.:-110.5               NA      : 11   01:03   :  7
 Max.   :-109.1               03:04   :  8   03:04   :  6
                              (Other): 15   (Other): 13
    ZMP          AML          ATPS         MP20
 01:01: 46   08:08   : 51   05:05   :155   05:07   : 64
 01:02: 51   07:07   : 42   03:03   : 69   07:07   : 53
 02:02:233   07:08   : 42   09:09   : 66   18:18   : 52
 NA   : 33   04:04   : 41   02:02   : 30   05:05   : 48
             NA      : 23   07:09   : 14   05:06   : 22
```

```
07:09  : 22   08:08  :  9   11:11  : 12
(Other):142   (Other): 20   (Other):112
```