An Introduction to stockR

Scott D. Foster
CSIRO, Hobart, Tasmania, Australia

Abstract

The stockR package is useful for finding stock structure within species (combined) populations. It does so by using a (potentially large) number of co-dominant genetic markers, for example SNPs. The package implements the methods described in Foster *et al.* (submitted), which is a variation on the models and computational methods described elsewhere (see reference list in Foster *et al.* submitted). In this tutorial, we will go through the steps of:

1. Simulate (from a mixture model) SNP data for a population of animals with a known number of stocks (sup-populations); and
2. Analyse the simulated data (and retrieve the simulated populations).

We shall perform this twice, once in the situation where none of the individuals are assumed to share the same stock and once where some individuals do. These individuals may have been sampled from a breeding aggregation (as in the yellowfin tuna data in Foster *et al.* submitted). This introduction is, largely because there is (at present) limited functionality in this package. However, we have concentrated on making the routines available to be *good* (we hope that the interface is fine too).

Keywords: Stock Structure, Sub-Population, SNP Marker, Mixture Model, R.

First Things First (setting up R for using stockR)

Before starting with this introduction to `stockR`, we need to make sure that everything is set up properly. Much of this will vary from computer to computer, but you must have a working version of R installed (preferably the latest one). This introduction was written and tested using R-3.4.0. It does not matter whether you prefer to use R through a development environment (such as RStudio) or through the command line – the results should be the same. So, start R and then:

```
install.packages( "stockR")
```

You will be asked which repository you want to use. Just use one that is geographically close to where you are (or where your computer is). Next load the package.

```
library( stockR)
```

For illustration is is also good to fix the random number seed, so that this document is reproducible *exactly*.

```
set.seed( 747)  #a 747 is a big plane
```

Now, we are good to go with the rest of the introduction.

Simulating a Co-Dominant Genetic Marker Data set

Let's simulate a data set, using the mixture model in Foster *et al.* (submitted, where the mathematical/technical treatment is also given). We shall simulate data from $K = 3$ stocks, without any sampling groups. It is important to note that the current implementation does not allow for specified stock sizes – the expectation of a stock size is the number of sampling groups divided by the number of stocks.

```
#number of individuals
N <- 100
#number of markers to measure on each of the individuals
M <- 5000
#number of sampling groups (same as number of individuals)
S <- N
#number of stocks
K <- 3
#simulate the data
myData <- sim.stock.data( nAnimal=N, nSNP=M, nSampleGrps=N, K=K)
```

This produces a data matrix of dimensions $(M \times N)$ so that the rows index the markers and the columns index the individuals. This data format is common to all functions within `stockR`. Although not highlighted here, it is possible to simulate datasets that have a limited number of informative markers – markers that are segregating between the stocks. Also, it is possible to alter the amount of separation between the stocks, but the default is demonstrated here (see Foster *et al.* submitted, for details and parameterisation).

We have now simulated data, so lets have a look at the contents of the data. Basically, the object `myData` is a matrix with the number of columns equal to the number of individuals and the number of rows equal to the number of loci (in this case it is a 5000×100 matrix). This object has a number of attributes though. These correspond to the stocks that each individual belongs to, as well as their sampling group. Ordering is the same as in the matrix of data too. Let's have a look at the data with code.

```
#the dimensions of the data
dim( myData)

[1] 5000  100

#the number of fish
ncol( myData)

[1] 100
```

```
#the number of markers
nrow( myData)

[1] 5000

#summary of the stock sizes
table( attributes( myData)$grps)


 1  2  3
20 77  3

#the individuals are ordered by stock membership
attributes( myData)$grps

  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [36] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [71] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3

#in this case the third stock is under-represented in the entire data set. On
#average, there will be equal numbers though.
```

The data themselves represent the number of copies of one of the alleles that each fish carries at each loci. Which allele is arbitrary, it could be (for example) SNP or reference. For exposition, let's call the two alleles A and B, and we shall store the data in terms of allele A. That is a 0 means zero copies (homozygous – BB), a 1 is heterozygous (AB and BA), and a 2 means 2 copies (homozygous – AA).

```
#a quick look at the data -- first 5 markers and 3 individuals
myData[1:5,1:3]

     [,1] [,2] [,3]
[1,]    0    1    0
[2,]    0    0    0
[3,]    2    2    2
[4,]    1    1    2
[5,]    1    1    1

#so, all individuals have two allele copies (homozygous) for the third loci
#the second fish is heterozygous for the allele at the first loci.
#and so on.
```

Finding Stocks In Simulated Data

The simulated data is designed to match the input requirements for the stock identification

function. For any real data set, there may have to be slightly more manipulation of the data. The key features of the input data is that: 1) it has as many rows as there are markers; 2) there are as many columns as individuals; 3) it is numeric (not a factor); 4) NAs are OK (they are ignored in grouping); and 5) the coding of data is by number of allele copies. The attributes of the simulated data (sample groups and stock membership) are not required to be present – although sample groups do need to be specified as their own function argument. Let's have a look at some code to find stocks:

```
#find the stocks in the data
stocks <- stockSTRUCTURE( myData, K=3)  #K-EM estimation is default
#the (posterior) membership probability to each stock
stocks$postProb #shows high discrimination, which could be erroneous (bootstrap soon)

    Pop_1 Pop_2 Pop_3
1       1     0     0
2       1     0     0
3       1     0     0
4       1     0     0
5       1     0     0
6       1     0     0
7       1     0     0
8       1     0     0
9       1     0     0
10      1     0     0
11      1     0     0
12      1     0     0
13      1     0     0
14      1     0     0
15      1     0     0
16      1     0     0
17      1     0     0
18      1     0     0
19      1     0     0
20      1     0     0
21      0     1     0
22      0     1     0
23      0     1     0
24      0     1     0
25      0     1     0
26      0     1     0
27      0     1     0
28      0     1     0
29      0     1     0
30      0     1     0
31      0     1     0
32      0     1     0
33      0     1     0
```

```
34      0       1       0
35      0       1       0
36      0       1       0
37      0       1       0
38      0       1       0
39      0       1       0
40      0       1       0
41      0       1       0
42      0       1       0
43      0       1       0
44      0       1       0
45      0       1       0
46      0       1       0
47      0       1       0
48      0       1       0
49      0       1       0
50      0       1       0
51      0       1       0
52      0       1       0
53      0       1       0
54      0       1       0
55      0       1       0
56      0       1       0
57      0       1       0
58      0       1       0
59      0       1       0
60      0       1       0
61      0       1       0
62      0       1       0
63      0       1       0
64      0       1       0
65      0       1       0
66      0       1       0
67      0       1       0
68      0       1       0
69      0       1       0
70      0       1       0
71      0       1       0
72      0       1       0
73      0       1       0
74      0       1       0
75      0       1       0
76      0       1       0
77      0       1       0
78      0       1       0
79      0       1       0
```

```
80      0     1     0
81      0     1     0
82      0     1     0
83      0     1     0
84      0     1     0
85      0     1     0
86      0     1     0
87      0     1     0
88      0     1     0
89      0     1     0
90      0     1     0
91      0     1     0
92      0     1     0
93      0     1     0
94      0     1     0
95      0     1     0
96      0     1     0
97      0     1     0
98      0     0     1
99      0     0     1
100     0     0     1

#hard classificaiton of individuals to stocks
stocks$hardClass <- apply( stocks$postProb, 1, which.max)
#Bootstrap to see about uncertainty in assignment
#for serious applications many more than B=25 will be needed.
#only 2 cores used to pass CRAN's arbitrary checks
stocks$boot <- stockBOOT( stocks, B=25, mc.cores=2)
#Assignment accounting for uncertainty.
#Could also look at other model quantities (e.g. allele frequencies)
stocks$uncertClass <- apply( stocks$boot$postProbs, 1:2, quantile, probs=c(0.05,0.5,0.95))
#careful inspection of this object will give the lower and upper 90% confidence in-
tervals and the median
#e.g. for the 99th individual
print( round( stocks$uncertClass[,99,], 3))

    Pop_1 Pop_2 Pop_3
5%  0.000     0     0
50% 0.000     0     1
95% 0.134     1     1

#showing that there is some uncertainty about which stock this indivudal belongs to.
```

There are a bunch of other output data in the `stocks` object. These either relate to the input data, the arguments used, or to potentially useful quantities (such as mean allele frequencies within each stock).

For these data, the stocks were reliably found. Care should be exerted though as there is no effort given to trying to account for label-switching of output (so estimated stock 1 may actually correspond to simulated stock 3, for example). The last lines of code assess the uncertainty in the assignment of the stock assignment. This is done by bootstrap methods (see Foster *et al.* submitted), and in this case we find 90% confidence intervals as well as the median. Results for only the 99^{th} individual are presented, which shows considerable variation – it is uncertain if this individual should be in the second or third stock.

The `stockSTRUCTURE` function has a number of different options for how the estimation of stock structure is performed. In most cases, the defaults should be reasonable but it is upon the user to make sure. Have a look at `?stockSTRUCTURE` and at Foster *et al.* (submitted) for the different options and what they implement.

Simulating and Finding Stocks with Sampling Groups

Code is now presented for simulating and analysing data with sampling groups, such as might arise from sampling a breeding aggregation. In this case, these samples are *a priori* known to come from the same stock and the analysis should assign all these fish to the same stock. See Foster *et al.* (submitted) for more details.

```
#number of sampling groups (same as number of individuals)
S <- 15   #there are now 15 sampling groups (100 individuals will be
        #distributed between them).
#number of stocks
K <- 3
#simulate the data
myData <- sim.stock.data( nAnimal=N, nSNP=M, nSampleGrps=S, K=K)
#find stocks in data
stocks <- stockSTRUCTURE( myData, K=3,
                         sample.grps=attributes(myData)$sampleGrps)
#once again, the stocks have been found (up to labelling)
#the simulated values
attributes( myData)$grps

 [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3

#the data-derived values (as hard clustered)
apply( stocks$postProbs, 1, which.max)

 samp1 samp10 samp11 samp12 samp13 samp14 samp15  samp2  samp3  samp4
     3      3      3      3      3      2      2      2      2      2
 samp5  samp6  samp7  samp8  samp9
     1      1      1      1      1

#Bootstrap to see about uncertainty in assignment
#for serious applications many more than B=25 will be needed.
#only 2 cores used to pass CRAN's arbitrary checks
```

```
stocks$boot <- stockBOOT( stocks, B=25, mc.cores=2)
#Assignment accounting for uncertainty.
#Could also look at other model quantities (e.g. allele frequencties)
stocks$uncertClass <- apply( stocks$boot$postProbs, 1:2, quantile, probs=c(0.05,0.5,0.95))
#careful inspection of this object will give the lower and upper 90% confidence in-
tervals and the median
#e.g. for the 99th individual
print( round( stocks$uncertClass, 3))

, , Pop_1

    samp1 samp10 samp11 samp12 samp13 samp14 samp15 samp2 samp3 samp4
5%      0      0      0      0      0      0      0     0     0     0
50%     0      0      0      0      0      0      0     0     0     0
95%     0      0      0      0      0      0      0     0     0     0
    samp5 samp6 samp7 samp8 samp9
5%      1     1     1     1     1
50%     1     1     1     1     1
95%     1     1     1     1     1

, , Pop_2

    samp1 samp10 samp11 samp12 samp13 samp14 samp15 samp2 samp3 samp4
5%      0      0      0      0      0      1      1     1     1     1
50%     0      0      0      0      0      1      1     1     1     1
95%     0      0      0      0      0      1      1     1     1     1
    samp5 samp6 samp7 samp8 samp9
5%      0     0     0     0     0
50%     0     0     0     0     0
95%     0     0     0     0     0

, , Pop_3

    samp1 samp10 samp11 samp12 samp13 samp14 samp15 samp2 samp3 samp4
5%      1      1      1      1      1      0      0     0     0     0
50%     1      1      1      1      1      0      0     0     0     0
95%     1      1      1      1      1      0      0     0     0     0
    samp5 samp6 samp7 samp8 samp9
5%      0     0     0     0     0
50%     0     0     0     0     0
95%     0     0     0     0     0

#Showing that there is almost no uncertainty about which stock this indivudal be-
longs to.
#There is lots of information in this data, especially when genetic information be-
tween
```

```
#individuals in the same sampling group is utilised.
```

Short Note on Missing Data

The simulated data above does not contain any missing marker scores, which will be ubiquitous in real data. However, `stockR` can handle missing data. It does so by assuming that the missing values are completely at random (e.g. missingness is not inheritable, nor is it related to stock structure). These values simply do not add to the likelihood of the individual belonging to any particular stock. Once again, see Foster *et al.* (submitted) for more detail. We cannot provide any guidelines to the user about the level of missingness one can tolerate in a marker's score (or an individual). Let us have a look at the behaviour of the code with some added missing values.

```
myData <- sim.stock.data( nAnimal=N, nSNP=M, nSampleGrps=N, K=K)
#add some missing data to simulation. There are 30% randomly missing scores.
totMark <- prod( dim( myData))
myData[sample( 1:totMark, size=floor( 0.3*totMark))] <- NA
#find stocks in data
stocks <- stockSTRUCTURE( myData, K=3)
#once again, the stocks have been found (up to labelling)
#the simulated values
attributes( myData)$grps
```

```
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 3 3 3 3
 [71] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
```

```
#the data-derived values (as hard clustered)
apply( stocks$postProbs, 1, which.max)
```

```
  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
  3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
 19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
  3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
 37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
  3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
 55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
  3   3   3   3   3   3   1   1   1   1   1   1   2   2   2   2   2   2
 73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
  2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
 91  92  93  94  95  96  97  98  99 100
  2   2   2   2   2   2   2   2   2   2
```

```
#bootstrap not performed for this example
```

Summary

In this short introduction the two functions in `stockR` have been introduced. These functions simulate data and analyse it for stock structure. They are easy to use and the default estimation method is fast (much faster than many other methods and slower than only one or two). See Foster *et al.* (submitted) for the comparison and for assessment of reliability.

If you have any queries regarding the package, then please contact the developer/maintainer. Your comments will be welcomed.

Last Things Last

The only remaining thing to do is to tidy up our workspace. This is just removing all objects for this analysis from your workspace. I like to do this, in tutorial situations, but you may not. It is entirely up to you whether you clean or not.

```
#tidy
rm( list=ls())
```

References

Foster SD, Feutry P, Grewe P, Berry O, Hui FKC, Davies C (submitted). "Reliably Discriminating Stock Structure With Genetic Markers: Mixture Models with Robust and Fast Computation."

1. Appendix

1.1. Computational details

This vignette was created using the following R and add-on package versions

- R version 3.4.3 (2017-11-30), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_AU.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_AU.UTF-8`, `LC_COLLATE=C`, `LC_MONETARY=en_AU.UTF-8`, `LC_MESSAGES=en_AU.UTF-8`, `LC_PAPER=en_AU.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_AU.UTF-8`, `LC_IDENTIFICATION=C`
- Running under: `Ubuntu 16.04.4 LTS`
- Matrix products: default
- BLAS: `/usr/lib/openblas-base/libblas.so.3`
- LAPACK: `/usr/lib/libopenblasp-r0.2.18.so`

- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: knitr 1.20, stockR 1.0.68
- Loaded via a namespace (and not attached): codetools 0.2-15, compiler 3.4.3, digest 0.6.15, evaluate 0.10.1, gtools 3.5.0, highr 0.6, magrittr 1.5, parallel 3.4.3, stringi 1.1.7, stringr 1.3.0, tools 3.4.3

Affiliation:

Scott D. Foster
CSIRO
Marine Laboratories
GPObox 1538
Hobart 7001
Australia E-mail: scott.foster@csiro.au