Mixed stock analysis in R: getting started with the `mixstock` package

Ben Bolker

September 1, 2014

1 Introduction

The `mixstock` package is a set of routines written in the R language (R Core Team, 2014) for doing mixed stock analysis using data on haploid genetic markers gathered from source populations and from one or more mixed populations. The package was developed for analyzing mitochondrial DNA (mtDNA) markers from sea turtle populations, but should be applicable to any case with discrete sources, discrete mixed populations, and discrete markers. (However, I do refer to sources as "rookeries" and markers as "haplotypes" throughout this document, and you will see other echoes of its origins, e.g. the number of markers is internally stored as variable `H` and the number of sources is stored as `R`.) The package is intended to be self-contained, but some familiarity with R or S-PLUS will definitely be helpful. (Some familiarity with your computer's operating system, which is probably Microsoft Windows, is also assumed.) The statistical methods implemented in the package are described in Bolker et al. (2003), Okuyama and Bolker (2005), Bolker et al. (2007), and Pella and Masuda (2001).

This package is in the public domain (GNU General Public License), is ©2008-2014 Ben Bolker and Toshinori Okuyama, and comes with NO WARRANTY. Please suggest improvements to me (Ben Bolker) at `bolker@mcmaster.ca`**.**

If you are feeling impatient and confident, turn to "Quick Start" (section 6).

2 Installation

You can skip this section if you are reading this file via the `vignette()` command in R— that means you've already successfully installed the package.

To get started, you will have to download and install the R package, a general-purpose statistics and graphics package, from CRAN (the "Comprehensive R Archive Network"); go to `http://www.r-project.org` and navigate from there[1]

The package has been developed under Linux and runs under Windows; it has been tested less thoroughly on MacOS. In general, the main compatibility/set-up issue will be running hierarchical models with WinBUGS or JAGS. The download file for R is about 52Mb. If you are running an antivirus package that is configured to check the signatures of executable files before they run, make sure you turn it off or register the new files installed by R before proceeding. You may also have some difficulty downloading packages if you have a firewall running on your computer — if you have trouble, you may want to (temporarily, at your own risk!) disable it.

Once you have downloaded and installed R, start the R program. The setup program should have asked whether you want to add a shortcut to the desktop or the Start menu: if you didn't, you will have to search for a file called `Rgui.exe`, which probably lives somewhere (on Windows) like `Program Files\R\R-3.1.1\bin` depending on what version of R you are using and where you decided to install it. R will open up a window for you with a command prompt (>), at which you can type R commands. (Don't panic.)

You can exit R by selecting `File/Exit` from the menus, or by typing `q()` at the command prompt. In general, if you want help on a particular command (e.g. `uml`) you can type a question mark followed by the command name (e.g. `?uml`)

You need to install the JAGS package (a separate software package that `mixstock` uses for some analyses). Go to http://sourceforge.net/projects/mcmc-jags/files/. The page should offer you a download compatible with your OS (look for "Looking for the latest version? Download JAGS-...")

If you plan to use WinBUGS (an alternative to JAGS that runs on Windows, and with some difficulty on MacOS or Linux), you will need to go to

[1]if you are in the US and using Windows, you can go directly to `http://cran.us.r-project.org/bin/windows/base/`: you will need to download a file called `R-x.y.z-win.exe` which will install R for you, when executed; `x.y.z` stands for the current version of R (3.1.1 as of September 1, 2014). Otherwise, see `http://www.r-project.org/mirrors.html` for a list of alternative "mirror sites" closer to you and navigate through the web pages to find a version to install (if you are not using Unix and/or an expert, you will want to look for a *binary* version of R).

the BUGS web page and follow the installation instructions.

You will next need to install the mixstock package and several other auxiliary packages, over the web, from within R. You will need to maintain a connection to the internet for this piece, although it is also possible to do this step off-line. Within R, at the command prompt, type

```
install.packages("mixstock")
```

this should install the package and all of the packages it depends on. If R asks you whether you want to delete the source files, answer y; you shouldn't need them again.

(If you don't have a convenient internet connection, you can download the files corresponding to mixstock and all of its dependent packages and install them by going to the Packages menu within R and choosing Install from local zip file. However, you will need compilation tools installed — see the documentation or FAQ for using R with your operating system.)

3 Loading the mixstock package and reading in data

Start every session with the mixstock package by typing

```
library(mixstock)

## Warning:  package 'mixstock' was built under R version
3.2.0
```

at the command prompt; this loads the mixstock package.

The package can read plain text data files that are separated by white space (spaces and/or tabs) or commas. If your data are in Microsoft Excel, you should export them as a comma-separated (CSV) file. If they are in Word, save them as plain text. The expected data format is that each row of data represents a haplotype, each column except the last represents samples from a particular rookery, and the last column is the samples from the mixed population. Each row and column should be named; your life will be simpler if the names do not have spaces or punctuation other than periods in them (a common R convention is to replace spaces with periods, e.g. North.FL for "North FL"). Do not label the haplotype column; R detects the presence of column names by checking whether the first row has one fewer item than the rest of the rows in the file.

For example, a plain text file (with haplotype labels `H1` and `H2` and rookery labels `R1`–`R3`) could look like this:

```
   R1  R2  R3  mix
H1  1  2  3  4
H2  3  4  5  6
```

Or a comma-separated file could look like this (note that the first line has only 4 elements while subsequent lines have 5).

```
R1,R2,R3,mix
H1,1,2,3,4
H2,3,4,5,6
```

If you have data from multiple mixed stocks, either put those data in a separate file or run them all together as columns of the same table (you will get a chance to specify how many sources and how many mixed populations there are):

```
R1,R2,R3,mix1,mix2
H1,1,2,3,4,7
H2,3,4,5,6,0
```

To read in your data, you first need to make sure that R knows how to find them. The easiest thing to do is to use the menu options[2] to move to a directory (i.e., folder) you will use for analysis, which should contain the data files you want to use and will contain R's working files. You can use the `getwd()` (get working directory) command to see where you are, and `list.files()` to list the files in the current directory. Once you have changed to the appropriate directory, you can read in your data files and assign the data to a variable. For example, if you had a file with space-separated data called `mydata.dat`, you could it read it in by typing

```
mydata = read.table("mydata.dat",header=TRUE)
```

and if you have a comma-separated file called `mydata.csv` you can use

[2]`File/Change working directory` on Windows, `Misc/Change working directory` or Apple-D on MacOS

```r
mydata = read.csv("mydata.csv")
```

(1) `header=TRUE` is required with `read.table` to specify that there is a header line in the file; it is part of the default settings for `read.csv`. Make sure there are no extra lines at the top of your data file, although you can use the `skip` argument (see `?read.table` for details) if necessary. (2) You must specify the *extension* of the file — the letters after the dot. Sometimes your operating system will hide that information from you.

If you have your own data you can read it in now and follow along, or you can use the `lahanas98raw` data set that comes with the package (Lahanas et al., 1998):

```r
mydata = lahanas98raw
```

To make sure that everything came out OK, type the name of the variable alone at the command prompt: e.g.

```r
mydata
```

to print out the data, or

```r
head(mydata)

##      FL MEXI CR AVES SURI BRAZ ASCE AFRI CYPR feed
## I    11    7  0    0    0    0    0    0    0    2
## II    1    0  0    0    0    0    0    0    0    0
## III  12    5 40    3    0    0    0    0    0   62
## IV    0    0  1    0    0    0    0    0    0    0
## V     0    1  0   27   13    0    0    0    0   10
## VI    0    0  0    0    1    0    0    0    0    0
```

to print out just the first few lines, as shown above.

Next, use the `as.mixstock.data` command to convert your data to a form that the package can use:

```r
mydata = as.mixstock.data(mydata)
```

Once your data are converted in this way, you can use `plot(mydata)` to produce a summary plot of the data (Figure 1).

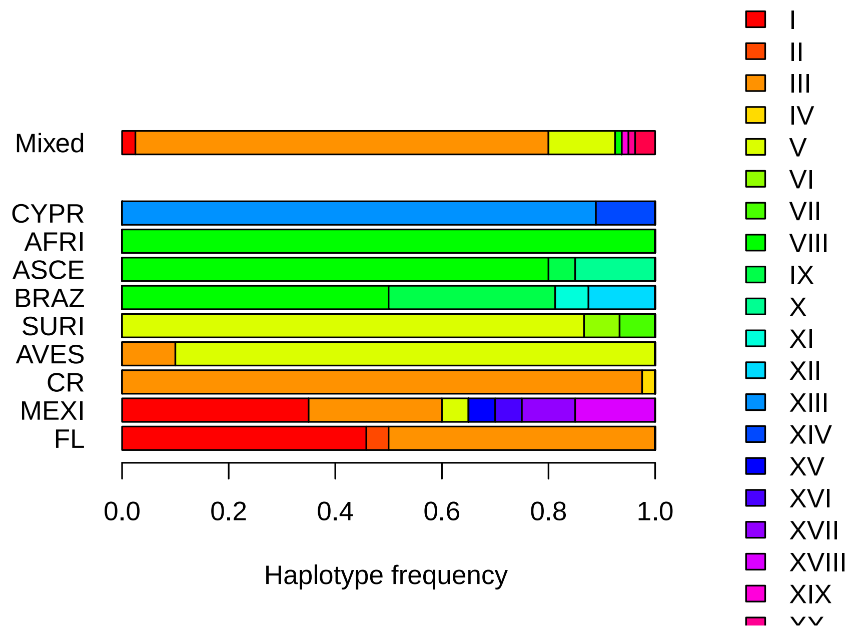


Figure 1: Basic plot of turtle mtDNA haplotype data, using `plot(mydata,mix.off=2)` (`mix.off=2` leaves a slightly larger space between the rookery and mixed stock data)

The default plot is a barplot, with the proportions of each haplotype sampled in each rookery represented by a separate bar; the mixed population data are shown as the rightmost bar.[3]

Before proceeding, you will need to "condense" your data set by (1) excluding any haplotype samples that are found only in the mixed population (such "singleton" haplotypes will break some estimation methods, and provide no useful information on turtle origins) and (2) lumping together all haplotypes that are found only in a single rookery and the mixed population (distinguishing among such haplotypes provides no extra information in our analyses, and may slow down estimation). You can do this by typing

```
mydata = markfreq.condense(mydata)
```

(To examine the condensed form of the data, you can print them by typing `mydata` at the command prompt, `head(mydata)` to see just the first few lines, or `plot(mydata)` to see the graphical summary [Figure 2].)

Some data are already entered in the package in the condensed format; `lahanas98` is the green turtle haplotype frequency data from Lahanas et al. (1998), while `bolten98` is the loggerhead haplotype frequency data from Bolten et al. (1998). Both of these data sets are already converted and condensed: `lahanas98raw` and `bolten98raw` give you the corresponding raw data tables.

```
## Warning:  some legend text may be truncated:  increase
leg.space?
```

4 Stock analysis

You can use the `mixstock` package to run various mixed-stock analyses on your data.

4.1 Conditional and unconditional maximum likelihood

You can do standard conditional maximum likelihood (CML) analysis using `cml(mydata)`. **to do: citations** If you want to save the results, you can save them as a variable that you can then print, plot, etc. (Figure 3)

[3]you can change from the default colors by specifying a `colors=` argument: e.g. if you have 10 haplotypes, `colors=topo.colors(10)` or `colors=gray((0:9)/9)`. See `?gray` or `?rainbow` for more information.

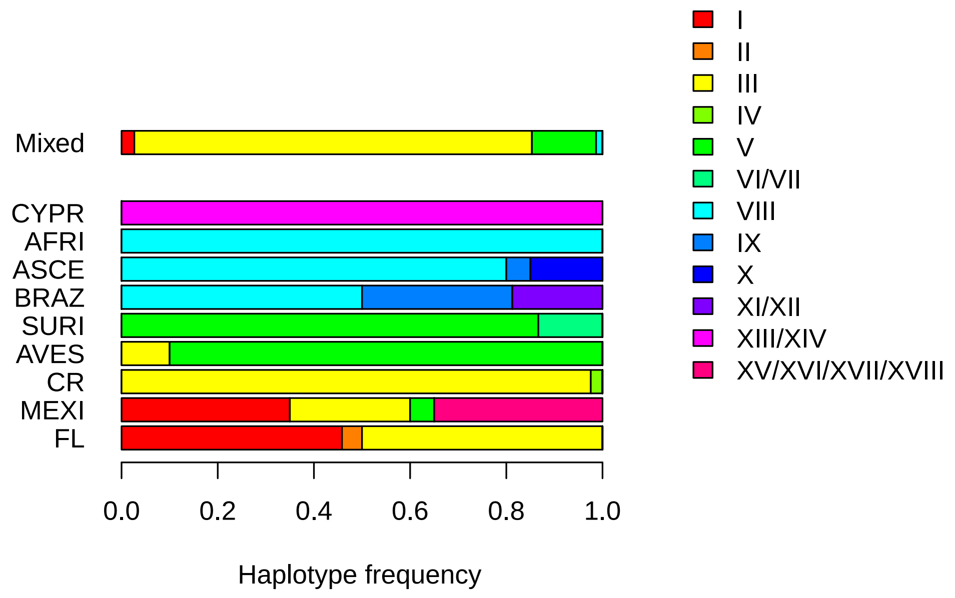


Figure 2: Condensed haplotype data from Lahanas 1998 (`plot(lahanas98, mix.off=2, leg.space=0.4)`; `leg.space=0.4` leaves more room for the legend).

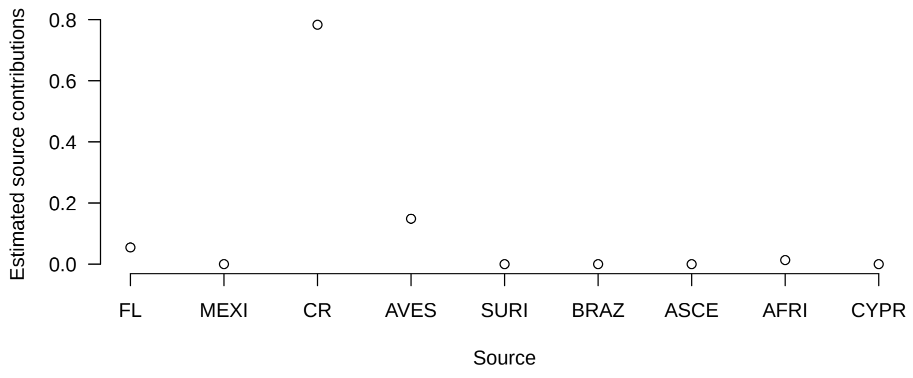


Figure 3: CML estimates for Lahanas 1998 data; `plot(mydata.cml)`

```
mydata.cml = cml(mydata)
mydata.cml

## Estimated input contributions:
##        FL      MEXI        CR      AVES      SURI      BRAZ      ASCE
## 5.463e-02 9.454e-05 7.834e-01 1.485e-01 1.333e-06 1.333e-06 1.333e-06
##      AFRI      CYPR
## 1.333e-02 1.333e-06
##
## Estimated marker frequencies in sources:
## (cml: no estimate)
##
## method: cml
```

Assigning the results of `cml` to a variable doesn't produce any output; you need to type the name of the variable to get the answers to print out.

Plotting the data produces a simple plot of the estimated contributions from each source (with no error bars): see Figure 3.

```
plot(mydata.cml)
```

When you print CML results, R will tell you there is no estimate for the rookery frequencies, because CML assumes that the true rookery frequencies are equal to the sample rookery frequencies, rather than estimating the rookery frequencies independently.

The default plot for estimation results plots points specifying the estimated proportions of the mixed population contributed by each rookery (to plot this with a logarithmic scale for the vertical axis, use `plot(mydata.cml,log="y")`).

Standard unconditional maximum likelihood analysis (UML) takes a little longer, but is equally straightforward (Smouse et al., 1990):

```
mydata.uml = uml(mydata)
```

UML estimates also include estimates of the true haplotype frequencies in each rookery, which are printed with the contribution estimates (as before, print these results by typing `mydata.uml` on a line by itself). As with CML, you can plot the results with `plot(mydata.uml)`; by default this plot includes just the rookery contribution information. You can include the estimated haplotype frequencies in the rookeries in the graphical summary as follows:

```
par(ask=TRUE)
plot(mydata.uml,plot.freqs=TRUE)
par(ask=FALSE)
```

(`par(ask=TRUE)` tells R to wait for user input between successive plots).

4.2 Confidence intervals: CML and UML bootstrapping

```
mydata.umlboot = genboot(mydata,"uml")
```

will generate standard (nonparametric) bootstrap confidence intervals for a UML fit to `mydata`, by resampling the data with replacement 1000 times (by default). *This is slow with a realistic size data set: it took 2.2 minutes to run 1000 bootstrap samples on my laptop.* (You can ignore warnings about `singular matrix, returning equal contribs`, `Error in qr.solve`, etc..) You can find out the results by typing

```
confint(mydata.umlboot)

##                          2.5%      97.5%
```

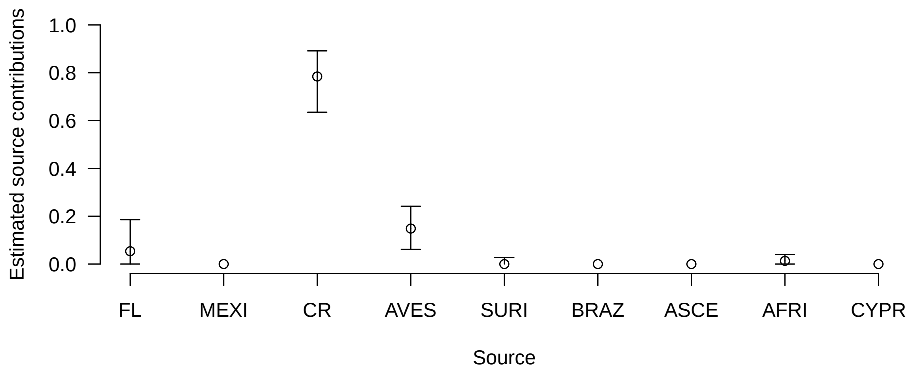


Figure 4: UML estimates with bootstrap confidence limits for Lahanas 1998 data: `plot(mydata.umlboot)`

```
## contrib.FL   1.000e-04 1.854e-01
## contrib.MEXI 8.256e-05 9.999e-05
## contrib.CR   6.350e-01 8.915e-01
## contrib.AVES 6.153e-02 2.417e-01
## contrib.SURI 1.080e-09 2.764e-02
## contrib.BRAZ 5.715e-10 1.845e-05
## contrib.ASCE 1.629e-13 3.672e-05
## contrib.AFRI 1.233e-13 4.000e-02
## contrib.CYPR 1.719e-13 2.408e-05
```

4.3 Markov Chain Monte Carlo estimation

```
mydata.mcmc = tmcmc(mydata)
```

```
mydata.mcmc

## Estimated input contributions:
##   contrib.FL contrib.MEXI   contrib.CR contrib.AVES contrib.SURI
##     0.055518     0.009707     0.777705     0.105770     0.036446
## contrib.BRAZ contrib.ASCE contrib.AFRI contrib.CYPR
##     0.003428     0.004219     0.005680     0.001527
##
## Estimated marker frequencies in sources:
## NULL
##
## method: mcmc
## prior strength: 0.1148
```

```
confint(mydata.mcmc)
```

```
##                     2.5%   97.5%
## contrib.FL   2.010e-11 0.23824
## contrib.MEXI 1.726e-17 0.07512
## contrib.CR   5.956e-01 0.89166
## contrib.AVES 3.616e-10 0.22609
## contrib.SURI 7.363e-16 0.17304
## contrib.BRAZ 1.665e-16 0.02786
## contrib.ASCE 8.068e-17 0.03001
## contrib.AFRI 3.821e-15 0.03643
## contrib.CYPR 9.119e-18 0.01507
```

```
plot(mydata.mcmc)
```

do the standard things: print the results, show confidence intervals, plot the results. (By default the information on haplotype frequencies in rookeries is not saved — it tends to be voluminous — and so this does not show up in the MCMC results.)

4.4 Convergence diagnostics for MCMC

When you are running MCMC analyses, you have to check that the Markov chains have *converged* (i.e. that you've run everything long enough for a reliable estimate).

4.4.1 Raftery and Lewis

The command

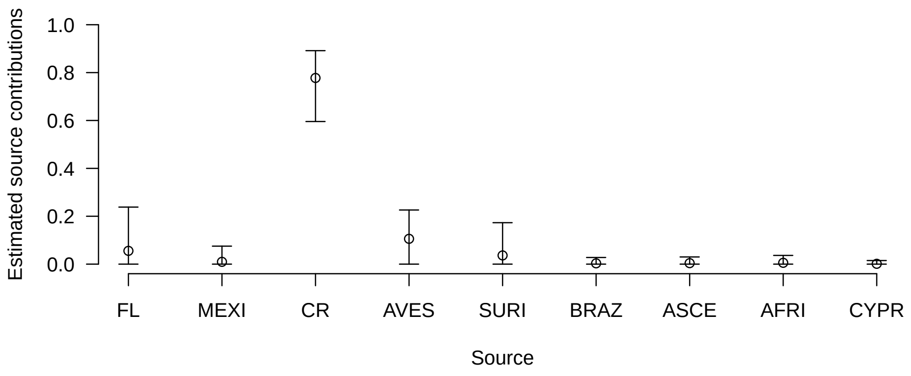


Figure 5: MCMC estimates with confidence limits for Lahanas 1998 data

```r
diag1=calc.RL.0(mydata)
```

(The final character is the numeral 0, not the letter O).

runs *Raftery and Lewis* diagnostics on your data set: these criteria attempt to determine how long a single chain has to be in order for it to give "sufficiently good" estimates. This function actually runs an iterative procedure, repeating the chain until the R&L criterion is satisfied.

The results consist of two parts:

- `diag1$current` gives the diagnostics for the last chain evaluated. These diagnostics consist of the predicted required length of the "burn-in" period (a transient that is discarded); the total number of iterations required; a lower bound on the total number required; and a "dependence factor" that tells how much correlation there is between subsequent values in the chain (see `?raftery.diag` for more information). Here are the first few lines of `diag1$current`:

```r
head(diag1$current)

##                Burn-in  Total  Lower bound  Dependence  factor
## contrib.FL          18   1521          235                6.47
## contrib.MEXI        14    926          235                3.94
## contrib.CR          28   1804          235                7.68
## contrib.AVES        4    312           235                   1.33
## contrib.SURI       15   1230           235                   5.23
## contrib.BRAZ        5    367           235                   1.56
```

- `diag1$suggested` gives the history of how long each suggested chain was as we went along: the iterations stop once suggested >current, but note that there is a lot of variability in the results.

```
diag1$history

##
## iteration Current Suggested
##         1     500       647
##         2     647      3882
##         3    3882      1804
```

4.4.2 Gelman and Rubin

The command

```
diag2=calc.GR(mydata)
```

tests the *Gelman-Rubin* criterion, which starts multiple chains from widely spaced starting points and tests to ensure that the chains "overlap" — i.e., that between-chain variance is small relative to within-chain variance. The general rule of thumb is that the criterion should be below 1.2 for all parameters in order for the chain to be judged to have converged properly. (Gelman et al., 1996).

5 Hierarchical models

To run hierarchical models, you will need to use either `WinBUGS` (on Windows, or on Linux or MacOS via a program called WINE, or some sort of Windows emulator) or `JAGS` (a newer, less well-tested program, but one that runs more easily on a variety of platforms).

Brief installation instructions for these programs:

- `WinBUGS`: go to `http://www.mrc-bsu.cam.ac.uk/bugs/winbugs/contents.shtml` and follow the instructions there to download and install `WinBUGS` version 1.4 and get a license key. Then make sure that you've installed the `R2WinBUGS` package (`install.packages("R2WinBUGS")`)
- `JAGS`: go to `http://www-fis.iarc.fr/~martyn/software/jags/` and download the appropriate version for your computer. Then install `R2jags` (`install.packages("R2jags")`)

You can use the `pm.wbugs()` command (with the same syntax as `tmcmc` above) to run basic mixed stock analysis (although `tmcmc` will in general be much more convenient and efficient: `pm.wbugs` is included for completeness and testing of `WinBUGS` methods). Use `mm.wbugs()` to run many-to-many analyses, with `R2WinBUGS` (default, `pkg="WinBUGS"`) or `JAGS` (`pkg="JAGS"`).

5.1 Many-to-many analysis

The `simmixstock2` command does basic simulation of multiple-mixed-stock systems. At its simplest, it simply generates random uniform values for the haplotype frequencies in each rookery and the proportional contributions of each rookery to each mixed stock (ref Figure 6).:

```
Z = simmixstock2(nsource=4,nmark=5,nmix=3,
                 sourcesize=c(4,2,1,1),
                 sourcesampsize=rep(25,4),
                 mixsampsize=rep(30,3),rseed=1001)
Z

## 4 sources, mixed stock(s), 5 distinct markers
## Sample data:
##    R1 R2 R3 R4 M1 M2 M3
## H1 14  8  5 14 12  6  9
## H2  2  0  0  0  4  3  1
## H3  2  2 11  5  4  6  3
## H4  2  2  7  0  4  6 11
## H5  5 13  2  6  6  9  6

plot(Z)
```

Now try to fit this via `mm.wbugs`:
Or, keeping the run in BUGS format for diagnostic purposes:

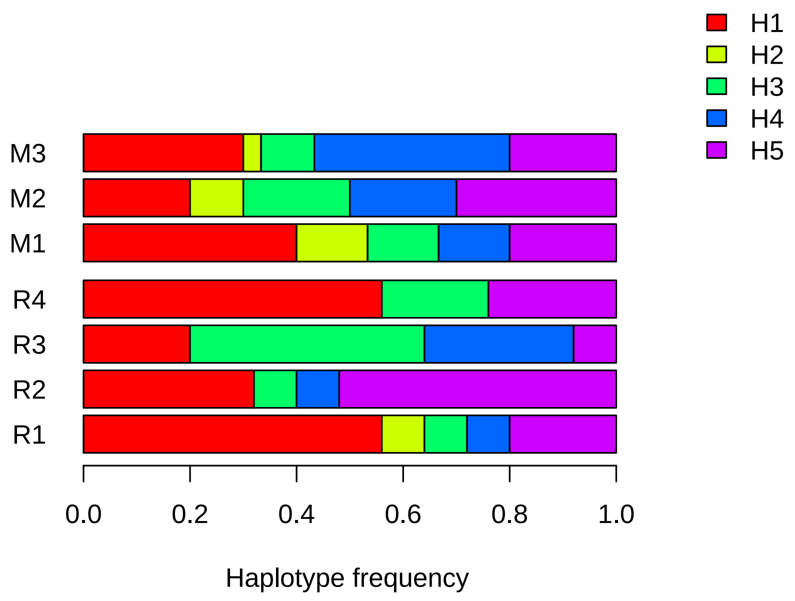


Figure 6: Simulated multiple-mixed-stock data

```
Zfit0 = mm.wbugs(Z,sourcesize=c(4,2,1,1),returntype="bugs")
```

This takes about 2.5 minutes to run with the default settings, which run 4 chains (equal to the number of sources) for 20,000 steps each. (There are two different versions of the BUGS code that can be used with `mm.wbugs`; in this particular case they give relatively similar answers and take about the same amount of time (`bugs.code="BB"` took 2.6 minutes), but if you're having trouble you might try switching from the default `bugs.code="TO"` to `bugs.code="BB"`.

Other important options when running `mm.wbugs` are:

- `n.iter`: the default is 20,000 iterations per chain, with the first half used as burn-in (`n.burnin=floor(n.iter/2)`); this may be conservative, and could take a long time with realistically large data sets. Use CODA's diagnostics as described above (`raftery.diag`, `gelman.diag`, etc.) to figure out an appropriate number of iterations.

- `n.chains`: equal to the number of sources by default, which may again be overkill. (Bolker et al. (2007) used three chains for an 11-source problem.)

- `inittype`: `"dispersed"` starts the chains from a starting point where 95% of the contributions are assumed to come from a single source; `"random"` starts the chains from random starting points. If `which.init` is specified, these sources will be used as the dominant starting points: for example, `mm.wbugs(...,n.chains=3,inittype="dispersed",whic` will start 3 chains with dominant contributions from sources 1, 5, and 7. If `which.init` is unspecified and `n.chains` is less than the number of sources, dominant sources will be picked at random.

- `returntype`: specifies what format to use for the answer. The default is a `mixstock.est` object that can be plotted or summarized like the results from any other mixed-stock analysis. However, for diagnostic purposes, it may be worth running the code initially with `returntype="bugs"` and using `as.mcmc.bugs` and `as.mixstock.est.bugs` to convert the result to either CODA format or mixstock format. Plotting bugs format and CODA format gives different diagnostic plots; CODA format can also be used to run convergence diagnostics such as `raftery.diag` or `gelman.diag`.

Plots from many-to-many runs:
Plot BUGS format diagnostics (plot not shown):

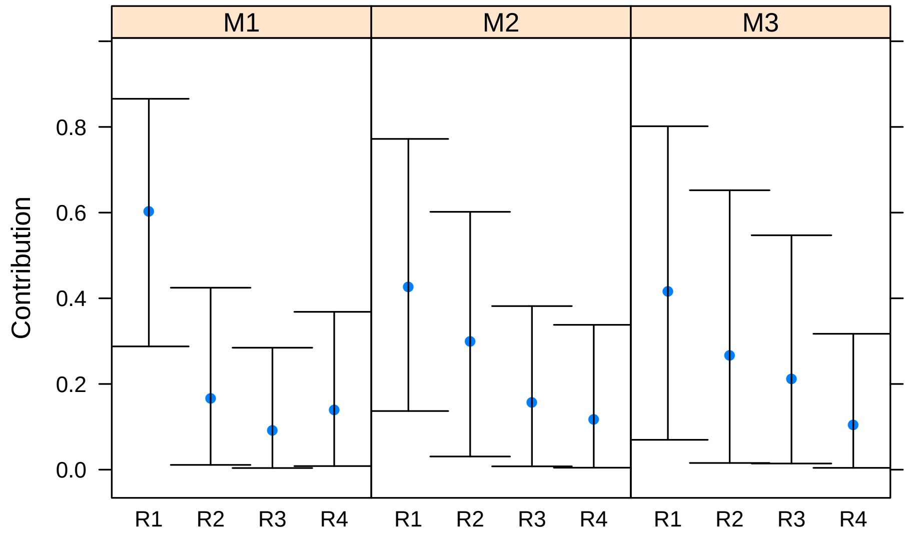
```
plot(Zfit0)
```

Plot CODA diagnostics (plot not shown):

```
plot(as.mcmc.bugs(Zfit0))
```

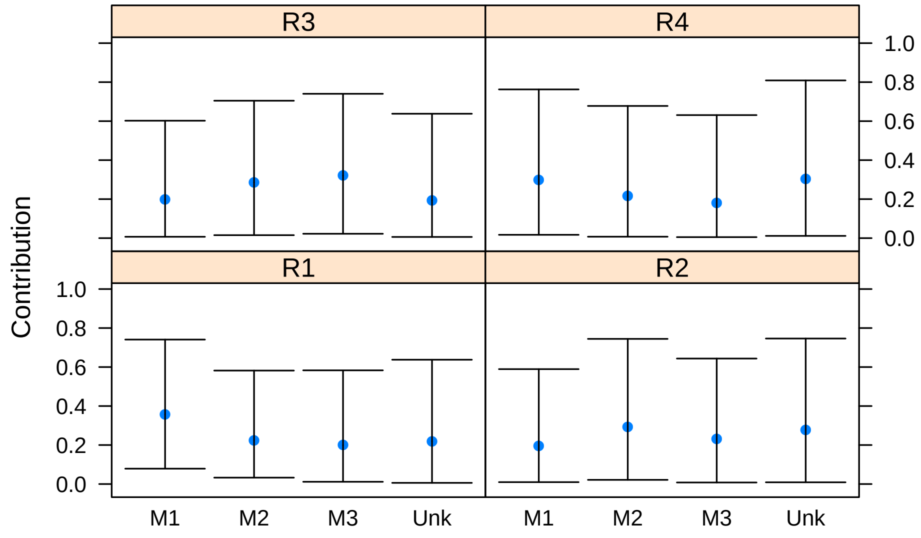
(the `plotMCMC` package generates prettier diagnostic output).
Plot results:

```
plot(Zfit)
```



Source-centric form:

```r
plot(Zfit,sourcectr=TRUE)
```



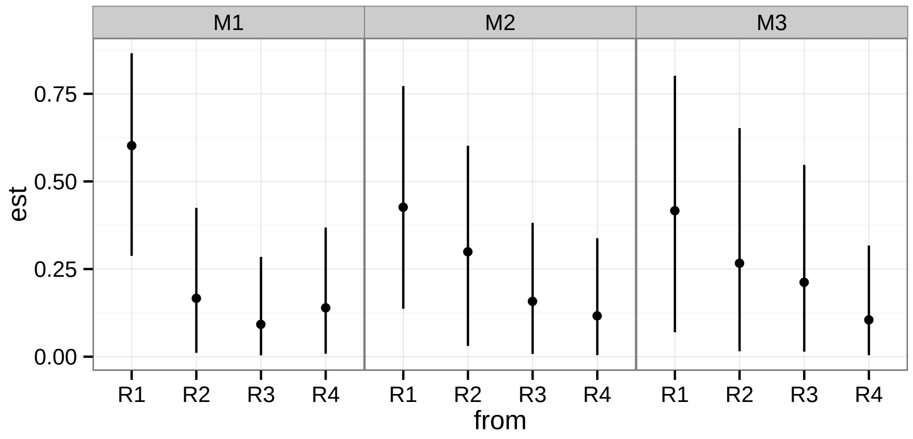
Summary/confidence intervals:

```
summary(Zfit)

## 4 sources, mixed stock(s), 5 distinct markers
## Sample data:
##    R1 R2 R3 R4 M1 M2 M3
## H1 14  8  5 14 12  6  9
## H2  2  0  0  0  4  3  1
## H3  2  2 11  5  4  6  3
## H4  2  2  7  0  4  6 11
## H5  5 13  2  6  6  9  6
##
## Estimates:
##
## Mixed-stock-centric:
##                  2.5%  97.5%
## R1.M1 0.6026 0.078822 0.7409
## R2.M1 0.1665 0.009677 0.5892
## R3.M1 0.0920 0.006879 0.6021
## R4.M1 0.1390 0.016973 0.7626
## R1.M2 0.4266 0.032832 0.5819
## R2.M2 0.2988 0.021459 0.7444
## R3.M2 0.1576 0.015216 0.7047
## R4.M2 0.1169 0.007170 0.6778
## R1.M3 0.4165 0.011549 0.5832
## R2.M3 0.2669 0.007883 0.6436
## R3.M3 0.2119 0.022271 0.7401
## R4.M3 0.1047 0.005100 0.6309
##
## Source-centric:
##                   2.5%  97.5%
## R1.Unk 0.3579 0.005933 0.6375
## R2.Unk 0.2226 0.008984 0.7462
## R3.Unk 0.1996 0.006015 0.6379
## R4.Unk 0.2199 0.011420 0.8087
## M1.R1  0.1955 0.287715 0.8657
## M2.R1  0.2939 0.136826 0.7720
## M3.R1  0.2322 0.069636 0.8016
## M1.R2  0.2784 0.011133 0.4247
## M2.R2  0.1989 0.030758 0.6018
## M3.R2  0.2860 0.015593 0.6522
## M1.R3  0.3206 0.003804 0.2846
## M2.R3  0.1945 0.007772 0.3818
## M3.R3  0.2989 0.014365 0.5471
## M1.R4  0.2177 0.008371 0.3684
## M2.R4  0.1802 0.004593 0.3380
## M3.R4  0.3033 0.004170 0.3172
```

Possibly prettier/do it yourself with ggplot:

```
Zfit.d <- as.data.frame(Zfit)
library(ggplot2)
library(grid)
theme_set(theme_bw())
zmargin <- theme(panel.margin=unit(0,"lines"))
ggplot(subset(Zfit.d,type=="input.freq"),
       aes(x=from,y=est,ymin=lwr,ymax=upr))+
    geom_pointrange()+facet_wrap(~to)+zmargin
```



6 Quick start

- Download and install R from CRAN (find the site closest to you at `http://cran.r-project.org/mirrors.html`; go to “Precompiled binary distributions” and from there to the base package; pick your operating system; download the setup program; and run the setup program).

- Download and install JAGS from http://sourceforge.net/projects/mcmc-jags/files/.
- Start R.
- From within R, download and install the `mixstock` package — this should automatically install all of the dependent packages as well.

  ```
  install.packages("mixstock")
  ```

  (This installation procedure needs to be done only once, although the `library` command below, loading the package, needs to be done for every new R session.)
- Load the package: `library(mixstock)`
- Load data from a comma-separated value (CSV) file, convert to proper format, and condense haplotypes:

  ```
  mydata = hapfreq.condense(as.mixstock.data(read.csv("myfile.dat")))
  ```
- analyze, e.g:

  ```
  mydata.mcmc = tmcmc(mydata)
  mydata.mcmc
  intervals(mydata.mcmc)
  plot(mydata.mcmc)
  ```

7 To do

- read.csv/read.table + as.mixstock.data combined into a single read.mixstock.data command? (also incorporate hapfreq.condense as a default option)
- `print.mixstock.est` could print sample frequencies instead of saying “no estimate” for CML

- MCMC section could be cleaned up considerably, explained better, R&L parameters not hard-coded, more efficient — don't re-run chains every time
- incorporate rookery sizes in data
- keep CODA objects or potential for CODA plots in MCMC results
- make MCMC convergence process more efficient: more explanation
- add hierarchical models????
- describe fuzz and bounds parameters on CML/UML, E-M algorithm
- plot(...,legend=TRUE) doesn't work for CML. add unstacked/beside=TRUE option to plot.mixstock.est
- incorporate source size data as part of data object
- some functions don't work with uncondensed data: fix or issue warning
- use `HPDinterval` from CODA for confidence intervals, rather than quantiles?

References

Bolker, B., T. Okuyama, K. Bjorndal, and A. Bolten (2003). Stock estimation for sea turtle populations using genetic markers: accounting for sampling error of rare genotypes. *Ecological Applications 13*(3), 763–775.

Bolker, B. M., T. Okuyama, K. A. Bjorndal, and A. B. Bolten (2007). Incorporating multiple mixed stocks in mixed stock analysis: 'many-to-many' analyses. *Molecular Ecology 16*, 685–695.

Bolten, A. B., K. A. Bjorndal, H. R. Martins, T. Dellinger, M. J. Biscotio, S. E. Encalada, and B. W. Bowen (1998). Transatlantic developmental migrations of loggerhead sea turtles demonstrated by mtDNA sequence analysis. *Ecological Applications 8*(1), 1–7.

Gelman, A., J. Carlin, H. S. Stern, and D. B. Rubin (1996). *Bayesian data analysis*. New York, New York, USA: Chapman and Hall.

Lahanas, P. N., K. A. Bjorndal, A. B. Bolten, S. E. Encalada, M. M. Miyamoto, R. A. Valverde, and B. W. Bowen (1998). Genetic composition of a green turtle (*Chelonia mydas*) feeding ground population: evidence for multiple origins. *Marine Biology 130*, 345–352.

Okuyama, T. and B. M. Bolker (2005). Combining genetic and ecological data to estimate sea turtle origins. *Ecological Applications 15*(1), 315–325.

Pella, J. and M. Masuda (2001). Bayesian methods for analysis of stock mixtures from genetic characters. *Fisheries Bulletin 99*, 151–167.

R Core Team (2014). *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. ISBN 3-900051-07-0.

Smouse, P. E., R. S. Waples, and J. A. Tworek (1990). A genetic mixture analysis for use with incomplete source population data. *Canadian Journal of Fisheries and Aquatic Sciences 47*, 620–634.