

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

Ben Bolker

March 20, 2012

1 Introduction

The `mixstock` package is a set of routines written in the R language R Development Core Team (2005) for doing mixed stock analysis using data on 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) and Pella and Masuda (2001).

This package is in the public domain (GNU General Public License), is ©2008 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.

26 To get started, you will have to download and install the R package,
27 a general-purpose statistics and graphics package, from CRAN (the “Com-
28 prehensive R Archive Network”); go to <http://www.r-project.org> and
29 navigate from there¹

30 The following installation instructions assume you are using a “modern”
31 Microsoft Windows system (tested on 2000 and XP); it is possible to use R,
32 and the `mixstock` package, on other operating systems — please contact the
33 authors for more information. (The package has been developed under Linux
34 and runs under Windows; most of it should run under MacOS as well, but it
35 is not as well supported and you will have to build the package from sources.
36 To run hierarchical models using WinBUGS, you need to have WINE set
37 up on Linux; I’m not sure about MacOS.) The setup file is about 17M,
38 and R takes up about 40M of disk space. If you are running an antivirus
39 package that is configured to check the signatures of executable files before
40 they run, make sure you turn it off or register the new files installed by R
41 before proceeding. You may also have some difficulty downloading packages
42 if you have a firewall running on your computer — if you have trouble, you
43 may want to (temporarily, at your own risk!) disable it.

44 Once you have downloaded and installed R, start the R program. The
45 setup program should have asked whether you want to add a shortcut to
46 the desktop or the Start menu: if you didn’t, you will have to search for
47 a file called `Rgui.exe`, which probably lives somewhere (on Windows) like
48 `Program Files\R\R-2.14.1\bin` depending on what version of R you are
49 using and where you decided to install it. R will open up a window for you
50 with a command prompt (`>`), at which you can type R commands. (Don’t
51 panic.)

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

56 You will next need to install the `mixstock` package and two other aux-
57 iliary packages, over the WWW, from within R (you will need to maintain
58 a connection to the internet for this piece, although it is also possible to do

¹if 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-win32.exe` which will install R for you, when executed; `x.y.z` stands for the current version of R (2.14.1 as of March 20, 2012). 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).

59 this step off-line). Within R, at the command prompt, type the following
60 commands:

```
> install.packages(c("mixstock", "plotrix", "coda", "abind", "R2WinBUGS"))
```

61 In each case, answer y to whether you want to delete the source files;
62 you shouldn't need them again.

63 (If you don't have a convenient internet connection, you can also down-
64 load the .zip files corresponding to the different packages and install them by
65 going to the **Packages** menu within R and choosing **Install from local**
66 **zip file**.)

67 **3 Loading the mixstock package and reading in** 68 **data**

69 Start every session with the `mixstock` package by typing

```
> library(mixstock)
```

70 at the command prompt; this loads the `mixstock` and auxiliary packages.

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

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

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

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

```

90 R1,R2,R3,mix
91 H1,1,2,3,4
92 H2,3,4,5,6

```

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

```

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

```

100 To read in your data, you first need to make sure that R knows how
 101 to find them. The easiest thing to do is to use the menu options² to move
 102 to a directory (i.e., folder) you will use for analysis, which should contain
 103 the data files you want to use and will contain R's working files. You can
 104 use the `getwd()` (get working directory) command to see where you are,
 105 and `list.files()` to list the files in the current directory. Once you have
 106 changed to the appropriate directory, you can read in your data files and
 107 assign the data to a variable. For example, if you had a file with space-
 108 separated data called `mydata.dat`, you could read it in by typing

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

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

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

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

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

```

> data(lahanas98raw)
> mydata = lahanas98raw

```

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

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

```
> mydata
```

121 to print out the data, or

```
> 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

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

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

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

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

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

130 Before proceeding, you will need to “condense” your data set by (1) ex-
 131 cluding any haplotype samples that are found only in the mixed population
 132 (such “singleton” haplotypes will break some estimation methods, and pro-
 133 vide no useful information on turtle origins) and (2) lumping together all
 134 haplotypes that are found only in a single rookery and the mixed population
 135 (distinguishing among such haplotypes provides no extra information in our
 136 analyses, and may slow down estimation). You can do this by typing

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

³you 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.

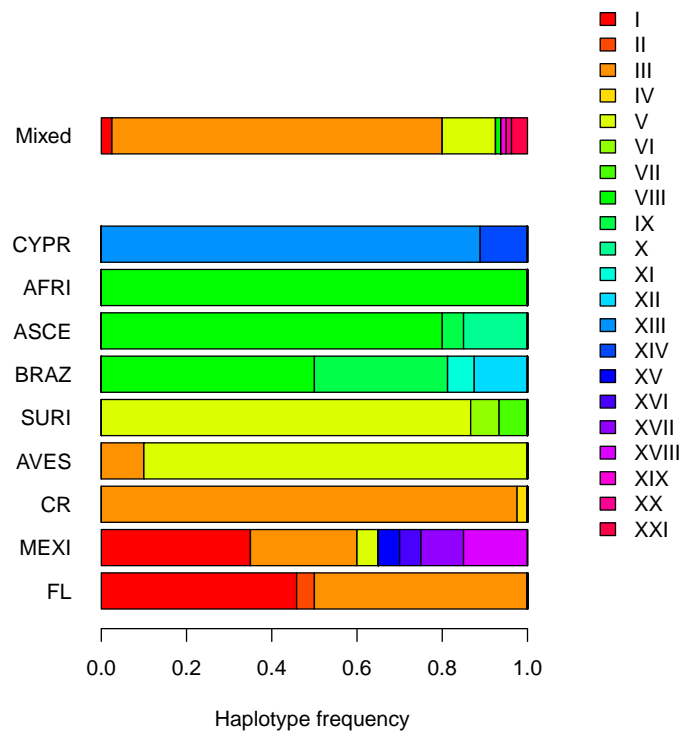


Figure 1: Basic plot of turtle mtDNA haplotype data, using `plot(mydata,mix.off=2)` (`mix.off=2` leaves a slightly larger space between the rookery and mixed stock data)

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

140 Some data are already entered in the package in the condensed format;
 141 you can access them using the `data()` command.

```
> data(lahanas98)
```

142 makes the haplotype frequency data from Lahanas et al. 1998 Lahanas et al.
 143 (1998) available as variable `lahanas98`, while

```
> data(bolten98)
```

144 makes the loggerhead data from Bolten et al. 1998 Bolten et al. (1998)
 145 available as `bolten98`, already converted and condensed: `bolten98raw` gives
 146 you the raw table.

147 4 Stock analysis

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

150 4.1 Conditional and unconditional maximum likelihood

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

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

Estimated input contributions:

	FL	MEXI	CR	AVES	SURI	BRAZ
	5.463021e-02	9.453698e-05	7.833919e-01	1.485493e-01	1.333410e-06	1.333277e-06
	ASCE	AFRI	CYPR			
	1.333144e-06	1.332877e-02	1.333010e-06			

Estimated marker frequencies in sources:
 (cml: no estimate)

method: cml

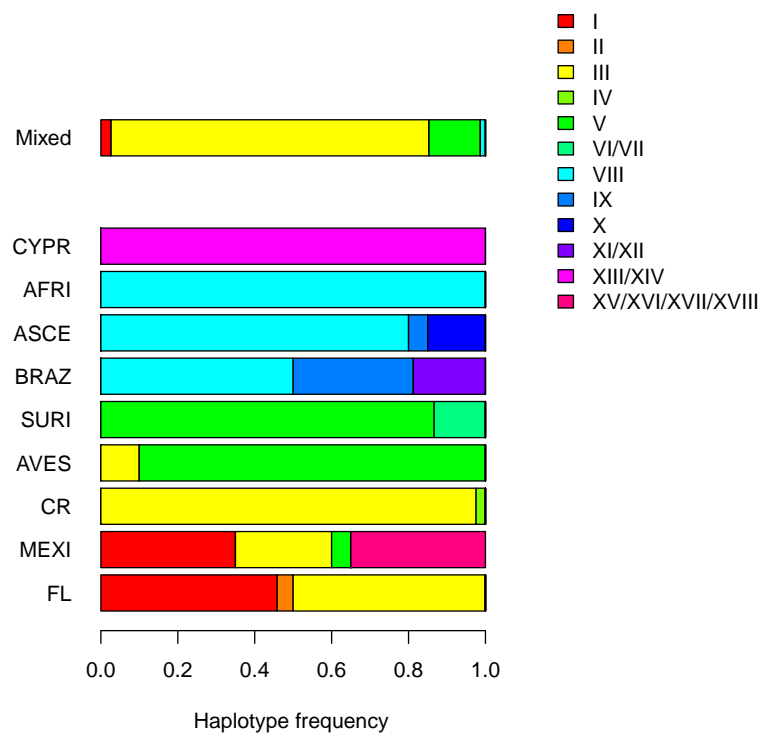


Figure 2: Condensed haplotype data from Lahanas 1998 (`plot(lahanas98, mix.off=2, leg.space=0.4)`; `leg.space=0.4` leaves more room for the legend)

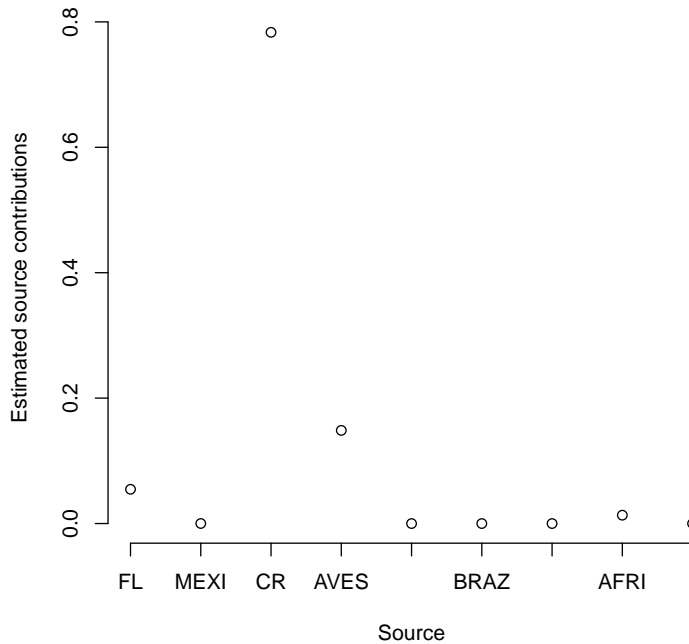


Figure 3: CML estimates for Lahanas 1998 data; `plot(mydata.cml)`

154 Assigning the results of `cml` to a variable doesn't produce any output;
 155 you need to type the name of the variable to get the answers to print out.
 156 Plotting the data produces a simple plot of the estimated contributions
 157 from each source (with no error bars): see Figure 3.

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

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

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

165 Standard unconditional maximum likelihood analysis (UML) takes a lit-
 166 tle longer, but is equally straightforward Smouse et al. (1990):

```

167 > mydata.uml = uml(mydata)
168
169 UML estimates also include estimates of the true haplotype frequencies
170 in each rookery, which are printed with the contribution estimates (as be-
171 fore, print these results by typing mydata.uml on a line by itself). As with
172 CML, you can plot the results with plot(mydata.uml); by default this plot
173 includes just the rookery contribution information. You can include the es-
174 timated haplotype frequencies in the rookeries in the graphical summary as
175 follows:

```

```

176 > par(ask=TRUE)
177 > plot(mydata.uml,plot.freqs=TRUE)
178 > par(ask=FALSE)

```

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

175 4.2 Confidence intervals: CML and UML bootstrapping

```

176 > mydata.umlboot = genboot(mydata,"uml")

```

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

```

182 > confint(mydata.umlboot)

```

	2.5%	97.5%
contrib.FL	1.000000e-04	1.853967e-01
contrib.MEXI	8.255739e-05	9.999000e-05
contrib.CR	6.349666e-01	8.915403e-01
contrib.AVES	6.152913e-02	2.417467e-01
contrib.SURI	1.079622e-09	2.764224e-02
contrib.BRAZ	5.715238e-10	1.844699e-05
contrib.ASCE	1.628700e-13	3.672277e-05
contrib.AFRI	1.232938e-13	3.999982e-02
contrib.CYPR	1.719070e-13	2.407764e-05

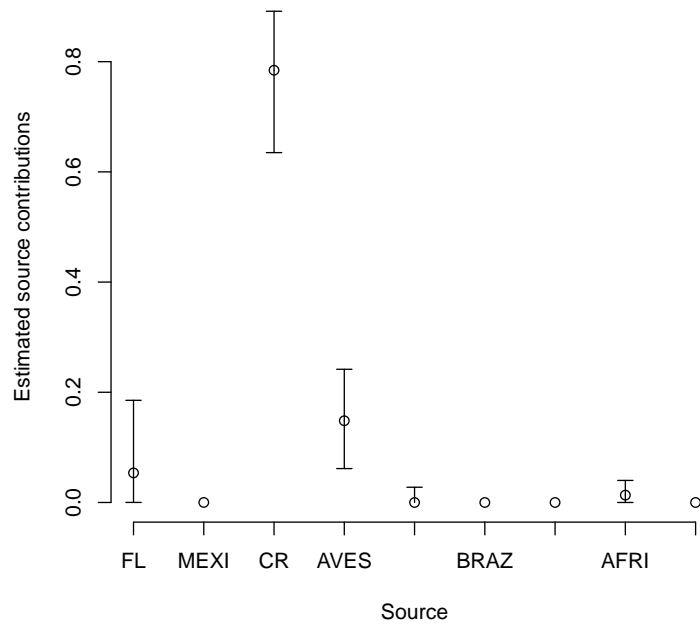


Figure 4: UML estimates with bootstrap confidence limits for Lahanas 1998
data: `plot(mydata.umlboot)`

182 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 contrib.BRAZ
0.055518267 0.009706668 0.777704826 0.105769897 0.036445990 0.003427765
contrib.ASCE contrib.AFRI contrib.CYPR
0.004219192 0.005680010 0.001527386

Estimated marker frequencies in sources:
NULL

method: mcmc
prior strength: 0.1147742

> confint(mydata.mcmc)

                2.5%      97.5%
contrib.FL  2.009853e-11 0.23823757
contrib.MEXI 1.726347e-17 0.07512486
contrib.CR   5.956080e-01 0.89165907
contrib.AVES 3.616006e-10 0.22608667
contrib.SURI 7.363441e-16 0.17303709
contrib.BRAZ 1.664703e-16 0.02785796
contrib.ASCE 8.067783e-17 0.03001117
contrib.AFRI 3.820586e-15 0.03642586
contrib.CYPR 9.118769e-18 0.01506706

> plot(mydata.mcmc)

```

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

187 4.4 Convergence diagnostics for MCMC

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

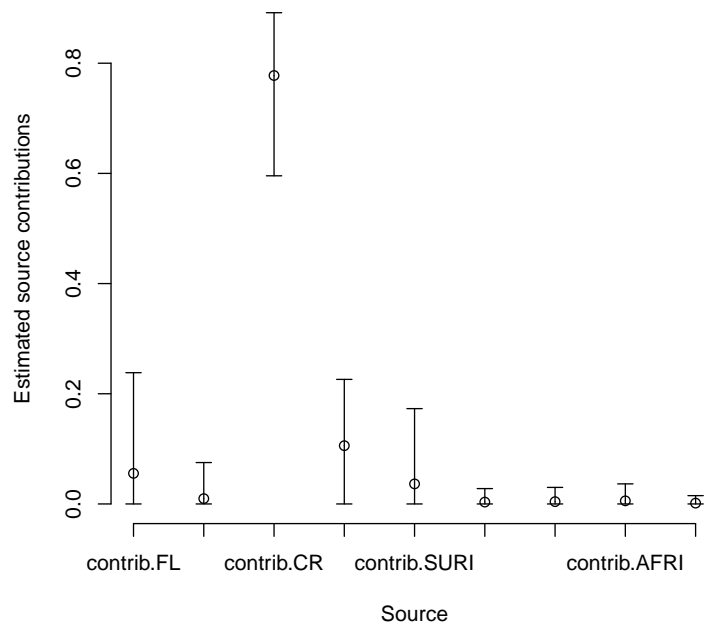


Figure 5: MCMC estimates with confidence limits for Lahanas 1998 data

191 4.4.1 Raftery and Lewis

192 The command

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

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

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

198 The results consist of two parts:

- 199 • **diag1\$current** gives the diagnostics for the last chain evaluated. These
200 diagnostics consist of the predicted required length of the “burn-in”
201 period (a transient that is discarded); the total number of iterations
202 required; a lower bound on the total number required; and a “dependence
203 factor” that tells how much correlation there is between subse-
204 quent values in the chain (see `?raftery.diag` for more information).
205 Here are the first few lines of **diag1\$current**:

```
> 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

- 206 • **diag1\$suggested** gives the history of how long each suggested chain
207 was as we went along: the iterations stop once suggested >current,
208 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

209 4.4.2 Gelman and Rubin

210 The command

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

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

217 5 Hierarchical models

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

222 Brief installation instructions for these programs:

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

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

235 5.1 Many-to-many analysis

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

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

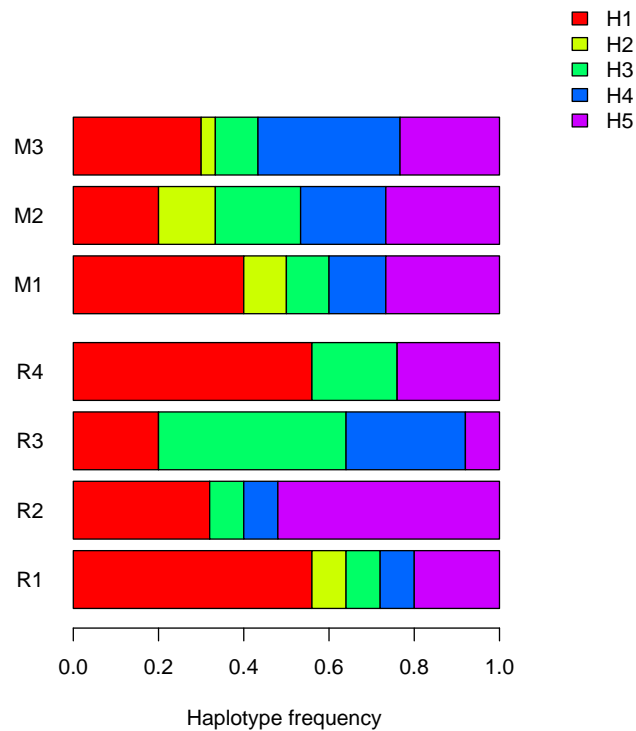
```
> Z
```

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	3	4	1
H3	2	2	11	5	3	6	3
H4	2	2	7	0	4	6	10
H5	5	13	2	6	8	8	7

```
> plot(Z)
```



240

241 Now try to fit this via `mm.wbugs`:

242 Or, keeping the run in BUGS format for diagnostic purposes:

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

243 This takes about 18.3 minutes to run with the default settings, which run
244 4 chains (equal to the number of sources) for 20,000 steps each. (There are
245 two different versions of the BUGS code that can be used with `mm.wbugs`;
246 in this particular case they give relatively similar answers and take about
247 the same amount of time (`bugs.code="BB"` took 9.2 minutes), but if you're
248 having trouble you might try switching from the default `bugs.code="T0"`
249 to `bugs.code="BB"`.

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

- 251 • **n.iter**: the default is 20,000 iterations per chain, with the first half
252 used as burn-in (`n.burnin=floor(n.iter/2)`); this may be conserva-
253 tive, and could take a long time with realistically large data sets. Use
254 CODA's diagnostics as described above (`raftery.diag`, `gelman.diag`,
255 etc.) to figure out an appropriate number of iterations.
- 256 • **n.chains**: equal to the number of sources by default, which may again
257 be overkill. (Bolker et al. (2007) used three chains for an 11-source
258 problem.)
- 259 • **inittype**: "dispersed" starts the chains from a starting point where
260 95% of the contributions are assumed to come from a single source;
261 "random" starts the chains from random starting points. If **which.init**
262 is specified, these sources will be used as the dominant starting points:
263 for example, `mm.wbugs(..., n.chains=3, inittype="dispersed", which.init=c(1,5,7))`
264 will start 3 chains with dominant contributions from sources 1, 5, and
265 7. If **which.init** is unspecified and **n.chains** is less than the number
266 of sources, dominant sources will be picked at random.
- 267 • **returntype**: specifies what format to use for the answer. The de-
268 fault is a `mixstock.est` object that can be plotted or summarized
269 like the results from any other mixed-stock analysis. However, for
270 diagnostic purposes, it may be worth running the code initially with
271 `returntype="bugs"` and using `as.mcmc.bugs` and `as.mixstock.est.bugs`
272 to convert the result to either CODA format or mixstock format. Plot-
273 ting bugs format and CODA format gives different diagnostic plots;
274 CODA format can also be used to run convergence diagnostics such as
275 `raftery.diag` or `gelman.diag`.

276 Plots from many-to-many runs:

277 Plot BUGS format diagnostics (plot not shown):

```

> plot(Zfit0)

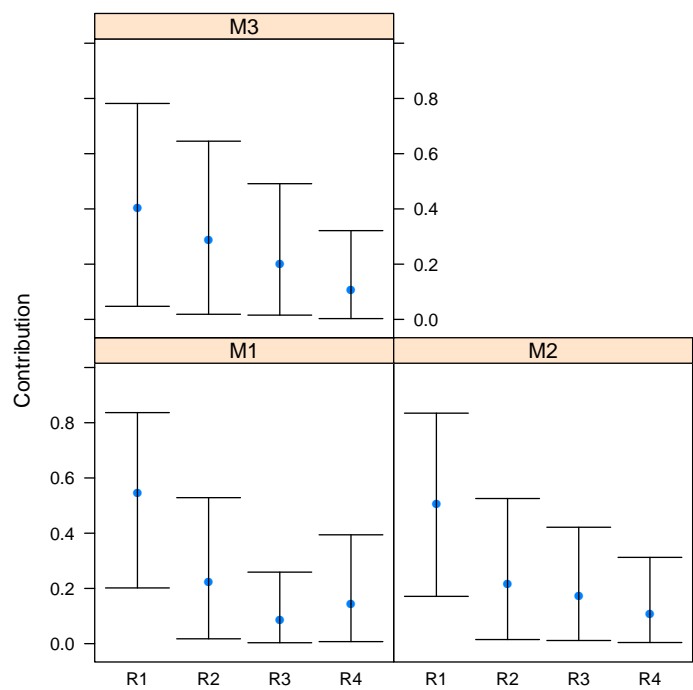
278   Plot CODA diagnostics (plot not shown):

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

279   Plot results:

> print(plot(Zfit))

```

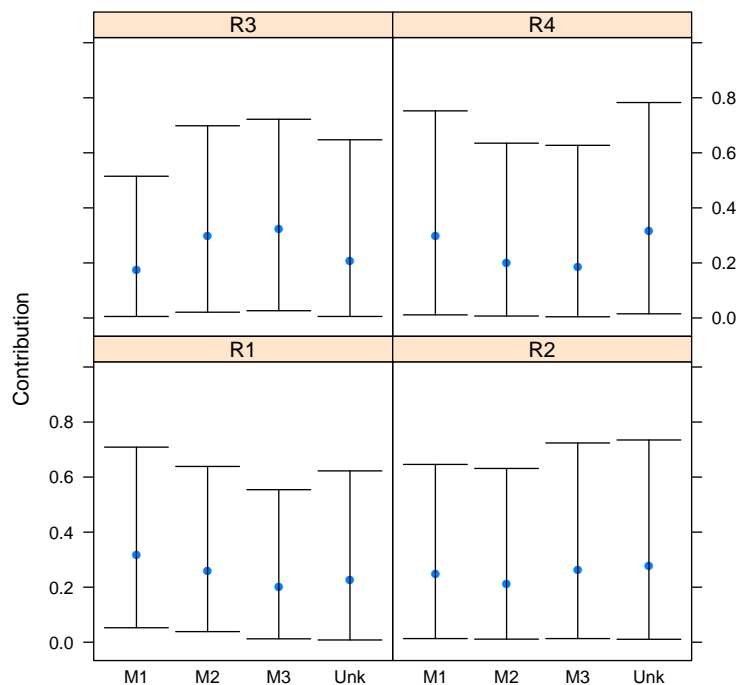


```

280   Source-centric form:

281   > print(plot(Zfit, sourcectr=TRUE))

```



282

283

Summary/confidence intervals:

```
> summary(Zfit)
```

4 sources, 3 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	3	4	1
H3	2	2	11	5	3	6	3
H4	2	2	7	0	4	6	10
H5	5	13	2	6	8	8	7

Estimates:

Mixed-stock-centric:

		2.5%	97.5%
M1.R1	0.5473780	0.201795000	0.8366150
M1.R2	0.2235784	0.017553250	0.5286050

```

M1.R3 0.0850429 0.003377650 0.2590050
M1.R4 0.1440014 0.007369775 0.3941075
M2.R1 0.5043251 0.171260000 0.8346125
M2.R2 0.2178163 0.014860500 0.5255300
M2.R3 0.1712309 0.011442625 0.4215025
M2.R4 0.1066277 0.004133800 0.3124100
M3.R1 0.4046099 0.047320750 0.7818925
M3.R2 0.2877887 0.018549000 0.6452925
M3.R3 0.2017308 0.015441500 0.4913425
M3.R4 0.1058681 0.002893225 0.3213625

```

Source-centric:

```

                2.5%    97.5%
R1.M1 0.3171615 0.052617250 0.7088300
R1.M2 0.2584727 0.038580500 0.6387150
R1.M3 0.1997042 0.012389250 0.5542900
R1.Unk 0.2246619 0.008175600 0.6225700
R2.M1 0.2492528 0.013269500 0.6460600
R2.M2 0.2118914 0.011240250 0.6314400
R2.M3 0.2626997 0.013295500 0.7239800
R2.Unk 0.2761556 0.010689750 0.7348300
R3.M1 0.1740109 0.005432050 0.5149200
R3.M2 0.2972163 0.020928500 0.6983675
R3.M3 0.3223322 0.026362250 0.7219875
R3.Unk 0.2064394 0.005509450 0.6473575
R4.M1 0.2988757 0.011309500 0.7524525
R4.M2 0.2004035 0.007036625 0.6351050
R4.M3 0.1847740 0.004338375 0.6272475
R4.Unk 0.3159484 0.015142750 0.7827350

```

284 (check this!)

285 6 Quick start

- 286 • Download and install R from CRAN (find the site closest to you at
287 <http://cran.r-project.org/mirrors.html>; go to “Precompiled bi-
288 nary distributions” and from there to the base package; pick your
289 operating system; download the setup program; and run the setup
290 program).

291 • Start R.

292 • From within R, download and install the `mixstock` package and aux-
293 iliary packages:

```
> install.packages("mixstock")  
> install.packages("plotrix")  
> install.packages("coda")  
> install.packages("abind")  
> install.packages("R2WinBUGS") ## or  
> install.packages("R2jags")
```

294 (This installation procedure needs to be done only once, although the
295 `library` command below, loading the package, needs to be done for
296 every new R session.)

297 • Load the package: `library(mixstock)`

298 • Load data from a comma-separated value (CSV) file, convert to proper
299 format, and condense haplotypes:

```
> mydata = hapfreq.condense(as.mixstock.data(read.csv("myfile.dat")))
```

300 • analyze, e.g.:

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

301 7 To do

302 • `read.csv/read.table + as.mixstock.data` combined into a single `read.mixstock.data`
303 command? (also incorporate `hapfreq.condense` as a default option)

304 • `print.mixstock.est` could print sample frequencies instead of saying
305 “no estimate” for CML

306 • MCMC section could be cleaned up considerably, explained better,
307 R&L parameters not hard-coded, more efficient — don’t re-run chains
308 every time

309 • incorporate rookery sizes in data

- 310 • keep CODA objects or potential for CODA plots in MCMC results
- 311 • make MCMC convergence process more efficient: more explanation
- 312 • add hierarchical models????
- 313 • describe fuzz and bounds parameters on CML/UML, E-M algorithm
- 314 • plot(...,legend=TRUE) doesn't work for CML. add unstacked/beside=TRUE
- 315 option to plot.mixstock.est
- 316 • incorporate source size data as part of data object
- 317 • some functions don't work with uncondensed data: fix or issue warning
- 318 • use `HPDinterval` from CODA for confidence intervals, rather than
- 319 quantiles?

320 References

- 321 Bolker, B., T. Okuyama, K. Bjorndal, and A. Bolten (2003). Stock esti-
 322 mation for sea turtle populations using genetic markers: accounting for
 323 sampling error of rare genotypes. *Ecological Applications* 13(3), 763–775.
- 324 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*. in press.
- 327 Bolten, A. B., K. A. Bjorndal, H. R. Martins, T. Dellinger, M. J. Biscotio,
 328 S. E. Encalada, and B. W. Bowen (1998). Transatlantic developmental
 329 migrations of loggerhead sea turtles demonstrated by mtDNA sequence
 330 analysis. *Ecological Applications* 8(1), 1–7.
- 331 Gelman, A., J. Carlin, H. S. Stern, and D. B. Rubin (1996). *Bayesian data*
 332 *analysis*. New York, New York, USA: Chapman and Hall.
- 333 Lahanas, P. N., K. A. Bjorndal, A. B. Bolten, S. E. Encalada, M. M.
 334 Miyamoto, R. A. Valverde, and B. W. Bowen (1998). Genetic composition
 335 of a green turtle (*Chelonia mydas*) feeding ground population: evidence
 336 for multiple origins. *Marine Biology* 130, 345–352.
- 337 Pella, J. and M. Masuda (2001). Bayesian methods for analysis of stock
 338 mixtures from genetic characters. *Fisheries Bulletin* 99, 151–167.

- 339 R Development Core Team (2005). *R: A language and environment for*
340 *statistical computing*. Vienna, Austria: R Foundation for Statistical Com-
341 puting. ISBN 3-900051-07-0.
- 342 Smouse, P. E., R. S. Waples, and J. A. Tworek (1990). A genetic mix-
343 ture analysis for use with incomplete source population data. *Canadian*
344 *Journal of Fisheries and Aquatic Sciences* 47, 620–634.