

# Package ‘mixstock’

January 2, 2012

**Version** 0.9.2

**Date** 2008/10/02

**Title** functions for mixed stock analysis

**Author** Ben Bolker <bolker@zoo.ufl.edu>

**Maintainer** Ben Bolker <bolker@zoo.ufl.edu>

**Depends** coda, plotrix, abind, lattice

**Suggests** R2WinBUGS, R2jags

**Description** mixed stock analysis functions

**License** GPL

**Repository** CRAN

**Repository/R-Forge/Project** mixstock

**Repository/R-Forge/Revision** 17

**Date/Publication** 2009-10-14 09:33:58

## R topics documented:

addlabels.barplot . . . . .	2
AIC.mixstock.est . . . . .	3
as.mcmc.bugs . . . . .	4
as.mixstock.data . . . . .	5
as.mixstock.est.bugs . . . . .	6
blockdiag . . . . .	7
bolten98 . . . . .	8
BUGS.out . . . . .	9
calc.GR . . . . .	9
calc.mult.GR . . . . .	10
calc.RL.0 . . . . .	11

cml . . . . .	13
coef.toptim . . . . .	14
expand.bugs.data . . . . .	15
genboot . . . . .	15
get.bot . . . . .	17
gibbsC . . . . .	18
gibbsmat . . . . .	19
hwdiag.check . . . . .	20
lahanas98 . . . . .	20
loglik2.C . . . . .	21
markfreq.condense . . . . .	22
marknames . . . . .	23
mcmc.chainlength.est . . . . .	24
mixstock.boot . . . . .	25
mixstock.est . . . . .	26
mm.wbugs . . . . .	27
mysum . . . . .	29
nmark . . . . .	29
normcols . . . . .	30
numberiv . . . . .	31
p.bayes . . . . .	31
packval . . . . .	32
plot.mixstock.est . . . . .	33
pm.wbugs . . . . .	34
q.to.p . . . . .	35
rdirichlet . . . . .	36
runsims . . . . .	37
simex . . . . .	38
simmixstock0 . . . . .	39
simmixstock1 . . . . .	40
simmixstock2 . . . . .	41
sourcesize.wbugs . . . . .	42
startvec0 . . . . .	42
tmcmc . . . . .	44
uml . . . . .	45
uml.lik . . . . .	47
<b>Index</b>	<b>49</b>

---

addlabels.barplot	<i>add labels to a barplot</i>
-------------------	--------------------------------

---

## Description

Adds labels (at specified heights) to an existing barplot

**Usage**

```
addlabels.barplot(x, vals, names, min = 0.1, cols = par("fg"), horiz = FALSE, ...)
```

**Arguments**

x	x positions (may be derived from a call to barplot)
vals	heights of labels
names	label text
min	minimum size for adding labels
cols	colors of label text
horiz	horizontal barplot
...	additional arguments to text()

**Value**

none.

**Author(s)**

Ben Bolker

**Examples**

```
set.seed(1001)
bvals <- matrix(runif(12),nrow=3)
b <- barplot(bvals)
addlabels.barplot(b,bvals,LETTERS[1:12])
```

---

AIC.mixstock.est      *commands for mixed stock analysis estimates*

---

**Description**

calculate AIC, confidence intervals, etc., for mixed stock analysis estimates

**Usage**

```
## S3 method for class 'mixstock.est'
AIC(object, ...)
## S3 method for class 'mixstock.est'
boxplot(x,...)
## S3 method for class 'mixstock.est'
coef(object,...)
## S3 method for class 'mixstock.est'
logLik(object,...)
## S3 method for class 'mixstock.est'
```

```

confint(object,parm,level=0.95,profile=FALSE,type=c("quantile","credible"),
show.sourcectr=TRUE,show.haps=FALSE,...)
## S3 method for class 'mixstock.est'
summary(object,...)
as.mixstock.est(object)

```

### Arguments

object	mixed stock analysis estimate
x	mixed stock analysis estimate
parm	parameter
level	confidence level
profile	confidence intervals by likelihood profiling? (stub)
type	for MCMC runs, produce credible intervals or quantiles of chains?
show.sourcectr	show confidence intervals for source-centric estimates?
show.haps	show confidence limits for marker frequencies?
...	additional arguments

### Value

Various summary statistics retrieved from mixed stock analysis estimates

### Note

AIC and logLik return statistical values etc. ... many of these are stubs

### Author(s)

Ben Bolker

---

as.mcmc.bugs

*convert from R2WinBUGS to CODA format*

---

### Description

converts output of R2WinBUGS to standard CODA format

### Usage

```

## S3 method for class 'bugs'
as.mcmc(x)

```

### Arguments

x	an object of class bugs, from the bugs function of R2WinBUGS
---	--

**Value**

an item of class `mcmc` or `mcmc.list`

**Author(s)**

Ben Bolker

---

as.mixstock.data      *class for marker data from sources and mixed stocks*

---

**Description**

This class provides a standard structure for information on markers (e.g. mitochondrial DNA samples) from a variety of different sources (e.g. sources) and from (a) mixed population(s) that draw(s) from those sources.

**Usage**

```
as.mixstock.data(object, nmix=1, sourcesize)
## S3 method for class 'mixstock.data'
plot(x, prop = TRUE, legend = TRUE, colors = rainbow(x$H),
     leg.space = 0.3, leg.ncol, leg.cex=1, mix.off = 0.5,
     stacklabels=FALSE, sampsize=FALSE, horiz=TRUE, ...)
## S3 method for class 'mixstock.data'
print(x, ...)
```

**Arguments**

object	a matrix with (R+1) columns and (H) rows, where the columns specify R sources plus a mixed population, and the rows specify distinct marker classes (e.g. mitochondrial haplotypes)
x	a <code>mixstock.data</code> object
prop	reduce data to frequencies?
legend	add a legend to the plot?
colors	colors denoting different markers
horiz	logical: plot bars horizontally?
leg.space	space to leave for legend (fraction at top of graph)
leg.ncol	number of columns for legend (default is 3 for horizontal barplots, 1 for vertical)
leg.cex	character size for legend
mix.off	spacing offset for bar(s) representing mixed stock(s)
stacklabels	(logical) put source names on multiple lines?
sampsize	(logical) add text showing sample sizes?

nmix	number of mixed stocks (default 1)
sourcesize	either a numeric vector of relative source sizes, or "first" or "last" to specify that the first or last row of the matrix contains the source sizes
...	additional arguments to <a href="#">barplot</a>

**Value**

mixstock.data objects have the following components:

R	number of sources
H	number of marker classes (haplotypes)
sourcesamp	samples from sources, in an HxR matrix
mixsamp	a vector of sample from the mixed population

print.mixstock.data and plot.mixstock.data give textual and graphical summaries of the results

**Note**

While vertical barplots are more familiar, horizontal barplots are useful for displaying long source/mixed stock names

**Author(s)**

Ben Bolker

**Examples**

```
x <- matrix(c(23,34,10,10,11,4,4,5,2),byrow=TRUE,nrow=3)
dx <- as.mixstock.data(x)
dx
plot(dx)
```

---

as.mixstock.est.bugs *Convert BUGS or CODA objects to mixstock estimate object*

---

**Description**

Converts output of a R2WinBUGS run, in bugs format or CODA (mcmc or mcmc.list) format, to a mixed-stock estimate object

**Usage**

```
as.mixstock.est.bugs(object,data=NULL,time)
```

**Arguments**

object	results of a BUGS run (from pm.wbugs etc.)
data	a mixstock.data object for filling in source and mixture names, etc.
time	time taken to run WinBUGS, in seconds

**Value**

an object of type mixstock.est

**Note**

as.mixstock.est.mcmc is a stub; for now, you can convert R2WinBUGS output to CODA or mixstock, but you can't do any other conversions

**Author(s)**

Ben Bolker

---

blockdiag                      *Block diagonal matrices*

---

**Description**

Construct a block-diagonal matrix from a list of submatrices.

**Usage**

```
blockdiag(...)
```

**Arguments**

...                      a comma-separated list of matrix objects that will form the submatrices of the block-diagonal matrix.

**Value**

A block-diagonal matrix with dimensions equal to the sum of the dimensions of the submatrices.

**Note**

If you want to use a previously constructed list of matrices, use `do.call` (see the examples). Doesn't preserve names or attributes (yet).

**Author(s)**

Ben Bolker

**Examples**

```
m1 <- matrix(1,nrow=2,ncol=2)
m2 <- matrix(2,nrow=1,ncol=3)
m3 <- matrix(3,nrow=3,ncol=2)
blockdiag(m1,m2,m3)
l <- list(m1,m2,m3)
do.call("blockdiag",l)
```

---

bolten98

*Loggerhead turtle data from Bolten 1998*

---

**Description**

Mitochondrial haplotype samples from Bolten 1998

**Usage**

```
data(bolten98)
```

**Format**

A data frame with 17 observations on the following 7 variables.

NWFL haplotype sample from northwest Florida

SOFL ditto, south Florida

NEFL.NC northeast Florida to North Carolina

Mexico Mexico

Greece Greece

Brazil Brazil

feed feeding grounds

**Source**

Bolten et al. 1998

**Examples**

```
data(bolten98)
```

---

BUGS.out	<i>generate MCMC output in BUGS format</i>
----------	--

---

**Description**

takes an MCMC object produced by `tmcmc` or `gibbsC` and sends it to an output file in BUGS output format, so that it can be read by CODA and other BUGS post-processors

**Usage**

```
BUGS.out(g, file=NULL)
```

**Arguments**

<code>g</code>	MCMC output as produced by <code>tmcmc</code>
<code>file</code>	output file: if NULL, uses the name of the object

**Value**

Sends output to a file.

**Author(s)**

Ben Bolker

---

<code>calc.GR</code>	<i>Run chains and calculate Gelman and Rubin diagnostics for mixed stock analyses</i>
----------------------	---

---

**Description**

For a mixed stock analysis data set with R sources, runs R Gibbs-sampler chains (one starting from a majority of the contribution from each source) and calculates Gelman and Rubin diagnostics on the combined set of chains.

**Usage**

```
calc.GR(data, tot=20000, burn=1, verbose=FALSE, rseed=1001,  
        chainfrac=NULL)
```

**Arguments**

data	A mixed stock analysis data set (either an object of type <code>mixstock.data</code> , or any list containing appropriate <code>sourcesamp</code> and <code>mixsamp</code> objects)
tot	Total length of each MCMC chain
burn	Burn-in time for MCMC chains
verbose	Produce verbose output?
chainfrac	Fraction of each chain to discard. By default, <code>chainfrac</code> is set to $(1-1/R)$ , so that the total length of the combined chain is equal to $(R*tot/R=tot)$ .
rseed	Seed for random-number generator

**Details**

calls CODA function `gelman.diag`

**Value**

Returns the diagnostic from the CODA function

**Author(s)**

Ben Bolker

---

calc.mult.GR	<i>Calculate Gelman and Rubin diagnostic multiple times for mixed stock analyses</i>
--------------	--

---

**Description**

Runs Gelman and Rubin diagnostics from CODA multiple times, to get an idea of the variation in convergence statistics

**Usage**

```
calc.mult.GR(data, n=10, tot=20000, burn=1, verbose=FALSE)
```

**Arguments**

data	Mixed stock analysis data (a list with elements <code>sourcesamp</code> and <code>mixsamp</code> )
n	Number of replicates to run
tot	Total number of iterates for each chain
burn	Burn-in time for each chain
verbose	Produce verbose output?

**Details**

Runs calc.GR multiple times, produces a summary table of the maximum point estimate and maximum 97.5% estimate (across all variables) for each run with a different random-number seed.

**Value**

A matrix with each row giving the random-number seed, max. point estimate, max. 97.5% quantile for each run.

**Note**

The generally accepted criteria for declaring convergence according to Gelman and Rubin is that all of the 97.5% quantiles of the estimates of shrink factors are less than 1.2.

**Author(s)**

Ben Bolker

**See Also**

calc.GR

**Examples**

```
data(simex)
calc.GR(simex, tot=2000)
```

---

calc.RL.0

*Use Raftery and Lewis diagnostics to calculate MCMC chain lengths*

---

**Description**

Uses coda::raftery.diag (the Raftery and Lewis diagnostic) to estimate minimum chain lengths for an MCMC estimate for mixed stock analysis. Runs R&L iteratively until the criteria are satisfied.

**Usage**

```
calc.RL.0(data, startfval, pilot=500, maxit=15, verbose=FALSE,
rseed=1001, debug=FALSE)
calc.mult.RL(data, n=50, debug=FALSE, verbose=FALSE)
RL.max(r)
```

**Arguments**

<code>data</code>	a <code>mixstock.data</code> object, or any list with <code>sourcesamp</code> and <code>mixsamp</code> entries containing source and mixed stock data
<code>startfval</code>	starting value of contribution frequencies for the chain, as in <code>gibbs</code> or <code>gibbsC</code> : <code>NULL</code> =random start, <code>0</code> =equal contributions from all sources, <code>(1..R-1)</code> =95% contribution from one source, with the rest splitting the remainder equally
<code>pilot</code>	Chain length to start with (length of "pilot" run)
<code>maxit</code>	Max. number of iterations of the Raftery and Lewis procedure
<code>verbose</code>	Produce lots of output?
<code>rseed</code>	Random-number seed
<code>debug</code>	produce debugging output?
<code>n</code>	number of different random-number seed chains to try
<code>r</code>	the results of a Raftery and Lewis diagnostic test

**Details**

`calc.RL.00` starts by running a Gibbs-sampler chain with the length given by `pilot`, then repeatedly lengthens the chain until the length is greater than that suggested as the total by the Raftery and Lewis diagnostic. (The next suggested step in the procedure is to run multiple chains of this length and see whether they pass the Gelman and Rubin diagnostic.) `calc.mult.RL` runs the Raftery and Lewis calculation multiple times, starting each chain from a large contribution from each source in turn, to see if some starting configurations are slower to converge or if there is a lot of variation among chains with different random number seeds. `RL.max` picks the expected maximum chain length given a set of diagnostics; `RL.burn` returns the predicted burn-in required.

**Value**

for `calc.RL.00`:

<code>current</code>	Results of the Raftery and Lewis test on the current iteration
<code>history</code>	History of the iterations:

for `calc.mult.RL`, a matrix giving the maximum expected chain length for each random-number seed/starting point combination

**Author(s)**

Ben Bolker

**Examples**

```
data(bolten98)
b98c <- markfreq.condense(as.mixstock.data(bolten98))
```

cml

*Mixed stock analysis by conditional maximum likelihood***Description**

Find the conditional maximum likelihood estimate (assuming marker frequencies in the sources are exactly equal to the sample frequencies) of the contributions of different sources to a mixed stock

**Usage**

```
cml(x, start.type="lsolve", fuzz=0, bounds=1e-4,
    ndepfac=1000, method="L-BFGS-B", lower=NULL, upper=NULL,
    ndeps=NULL,
    control=NULL, debug=FALSE, transf="part", grad=cml.grad, ...)
```

**Arguments**

x	a list with elements <code>mixsamp</code> (a vector of the sampled markers in the mixed stock) and <code>sourcesamp</code> (a matrix, with markers in rows and sources in columns, of markers in the source samples)
grad	function giving the gradient of the likelihood
start.type	starting values to use: <code>equal</code> (equal contributions from each source); <code>random</code> (multinomial sample with equal probabilities); <code>rand2</code> (sample out of a transformed normal distribution); a number between 1 and the number of sources; that source starts with 0.95 contribution and the rest start with 0.05/(R-1); default <code>lsolve</code> , the linear solution to the problem
fuzz	min. value (1-min is the max.) for starting contributions
bounds	( <code>bounds</code> , <code>1-bounds</code> ) are the lower and upper bounds for mle calculations
ndepfac	factor for computing numerical derivatives; numerical derivative stepsize is computed as <code>bounds/ndepfac</code> [OBSOLETE with gradient function?]
method	optimization method, to be passed to <code>optim</code>
debug	produce debugging output?
lower	lower bound
upper	upper bound
ndeps	scaling factor for optimization
control	other control arguments to <code>optim</code>
transf	(character) <code>"full"</code> : use arctan transform to transform $(-\infty, \infty)$ to $(0, 1)$ or vice versa; <code>"part"</code> : don't; <code>"none"</code> : no transform
...	other arguments to <code>mle</code> or <code>optim</code> (e.g. <code>hessian=FALSE</code> to suppress (slow) hessian calculation, etc.)

**Details**

By default, uses `mle` which in turn uses `optim` with `method="L-BFGS-B"` to do bounded optimization

**Value**

an object of class `mixstock.est`, containing the results of the fit

**Author(s)**

Ben Bolker

**Examples**

```
true.freq <- matrix(c(0.65,0.33,0.01,0.01,
                    0.33,0.65,0.01,0.01),ncol=2)
true.contrib <- c(0.9,0.1)
x <- simmixstock0(true.freq,true.contrib,50,100,1004)
cml.est <- cml(x)
cml.est
```

---

coef.toptim

*minimal structure for optim fits*

---

**Description**

provides `coef()` and `logLik()` methods for `optim` fits

**Usage**

```
## S3 method for class 'toptim'
coef(object,...)
## S3 method for class 'toptim'
logLik(object,...)
```

**Arguments**

`object` the result of an `optim` fit  
`...` optional arguments (for generic compatibility)

**Details**

extracts `par` and `-val` respectively

**Value**

coefficients or value

**Note**

mle is not flexible enough to take vectors as inputs ...

---

expand.bugs.data	<i>convert data to BUGS input format</i>
------------------	--

---

**Description**

expands mixed stock data to BUGS input format (indicator variables)

**Usage**

```
expand.bugs.data(mixsamp)
```

**Arguments**

mixsamp	mixed-sample or mixed stock data matrix (numbers from each source population)
---------	---

**Value**

a matrix of indicator (0/1) variables: number of rows = total number of individuals, number of columns = number of source populations

**Author(s)**

Ben Bolker

**Examples**

```
expand.bugs.data(c(A=4,B=2,C=5))
```

---

genboot	<i>Generate bootstrap estimates of mixed stock analyses</i>
---------	---

---

**Description**

Given a data set or simulated data set (in the usual format of a list of source samples and samples from the mixed stock), generate bootstrap-resampled data sets (either parametrically or nonparametrically) and a vector of estimates from the resampled data sets

**Usage**

```
genboot(x, method="cml", nboot=1000, rseed=1001, verbose=FALSE,
fuzz=0.001, maxfail=100, rpt=20, start.type="lsolve",
param=FALSE,param.match="mean", ndepfac=10000, save.boot=FALSE,
print.boot=FALSE,...)
```

**Arguments**

x	A mixed stock data set: a list with elements <code>sourcesamp</code> (a matrix, with sources as columns and markers as rows, of samples in the sources) and <code>mixsamp</code> (a vector of markers sampled in the mixed stock)
method	"cml" for conditional max. likelihood or "uml" for unconditional max. likelihood
nboot	number of bootstrap samples
rseed	random-number seed
verbose	produce verbose output?
fuzz	small value for keeping estimated frequencies away from 0/1
maxfail	number of consecutive fitting tries before entering NAs for a given simulation
rpt	frequency with which to report progress
start.type	starting conditions for fitting procedure (see <a href="#">startvec</a> and/or <a href="#">startvec0</a> )
param	Do parametric bootstrapping?
param.match	match mean or mode of resampled frequencies?
ndepfac	scaling factor for CML fit
save.boot	save bootstrap replicates?
print.boot	verbose output from bootstrap?
...	additional arguments to <a href="#">cml</a>

**Value**

An object of type `mixstock.est` with element `resample` as a matrix of bootstrap results (where the columns are the parameters, the negative log-likelihood of the fit, and a code for convergence).

**Examples**

```
data(simex)
x <- genboot(simex,method="cml",nboot=100)
r <- x$resample
r.ok <- r[!is.na(r[,"Convergence"]) & r[,"Convergence"]==0,]
old.par <- par(no.readonly=TRUE)
par(mfrow=c(2,2))
hist(r.ok[,1],main="Contrib. A")
hist(r.ok[,2],main="Contrib. B")
hist(r.ok[,3],main="Neg. log likelihoods")
plot(r.ok[,1],r.ok[,3],
      xlab="Contrib. A",ylab="NLL")
par(old.par)
```



a	prior strength
startiter	starting iteration
maxiter	maximum iteration
startfval	starting fvalue
thin	thinning factor
fprior	source prior
rptiter	how often to print output to the screen

**Note**

put.mp and get.mp.input are portmanteau functions

**Author(s)**

Ben Bolker

---

gibbsC

*Run mixed stock analysis Gibbs sampler, in C*

---

**Description**

Runs a Gibbs sampler MCMC for mixed stock analysis, calling a routine written in C (for code). Low-level function, called by other functions.

**Usage**

```
gibbsC(a=1, startiter, maxiter, data, mixsamp=NULL, sourcesamp=NULL,
startfval=NULL, thin=1, fprior=NULL, outfile=FALSE,
outfn="mixstock-gibbs", randseed=1001, rptiter=-1, debug=FALSE,
contrun=FALSE, contrib.start=NULL, sourcefreq.start=NULL)
```

**Arguments**

a	Prior strength parameter
startiter	Number of iterations to discard (burn-in)
maxiter	Total number of chain steps
data	A mixstock.data object
mixsamp	Marker sample from mixed populations
sourcesamp	Marker samples from sources
startfval	Where to start the chain: 0=
thin	thinning factor
fprior	Bayesian prior
outfile	send data to an output file?

outfn	name of output file
randseed	random-number seed
rptiter	frequency for sending reports to screen
debug	debug?
contrun	continuation run? set fval and sourcefreq directly
contrib.start	vector of starting contributions
sourcefreq.start	matrix of starting source freqs

**Value**

a numeric matrix containing samples from the chain: each row is a vector of estimated contribution frequencies from each rookery

**Note**

gibbsC calls C code to generate multinomial deviates derived from the randlib.c library (version 1.3: <http://odin.mdacc.tmc.edu/anonftp/>), written by Barry W. Brown, James Lovato, Kathy Russell, and John Venier, derived in turn from page 559 of: Devroye, Luc, Non-Uniform Random Variate Generation. Springer-Verlag, New York, 1986.

**Author(s)**

Ben Bolker

---

gibbsmat	<i>Plot Gibbs sampler output</i>
----------	----------------------------------

---

**Description**

plot or report on output from Gibbs sampler

**Usage**

```
gibbsmat(x, burnin = 500, R = 2, H = 2, trans = TRUE)
gibbsrpt(x, burnin=500, R=2, H=2, trans=TRUE, plot=TRUE)
plotvar(vec, best, name, ...)
```

**Arguments**

x	results of an MCMC fit
burnin	length of burn-in period for chain
R	number of rookeries
H	number of haplotypes
trans	transform chain values?

plot	plot posterior distributions for each variable?
vec	numeric vector (posterior distribution chain)
best	best-fit/point estimate value
name	name of variable, for x label
...	other arguments to plot

**Value**

sets up an array of plots on the current output device

---

hwdiag.check	<i>check Heidelberger and Welch diagnostic results</i>
--------------	--

---

**Description**

Runs the Heidelberger and Welch tests (`hwdiag`) implemented in coda and tests that all results are OK

**Usage**

```
hwdiag.check(x, verbose = FALSE)
```

**Arguments**

x	the results of a model fit?
verbose	verbose output?

**Value**

logical: true or false

---

lahanas98	<i>Green turtle data from Lahanas et al. 1998</i>
-----------	---

---

**Description**

This data set gives counts of different mitochondrial haplotypes sampled in different rookeries throughout the Atlantic and Mediterranean (Florida (FL), Mexico (MEXI), Costa Rica (CR), Aves Island (AVES), Surinam (SURI), Brazil (BRAZ), Ascension Island (ASCE), Africa (AFRI), Cyprus (CYPR)), as well as in a pooled feeding-ground population.

**Usage**

```
data(lahanas98)
```

**Format**

(lahanas98) a mixstock.data object; (lahanas98raw) a text table with 10 columns (9 rookeries + feeding ground) and 21 rows (haplotypes)

**Source**

Lahanas et al. 1998

---

loglik2.C	<i>likelihood calculation, in C</i>
-----------	-------------------------------------

---

**Description**

takes a parameter vector corresponding to the (possibly transformed) source contributions and source marker frequencies and returns the negative log likelihood

**Usage**

```
loglik2.C(p, sourcesamp, mixsamp = NULL, verbose = FALSE, transf = c("full", "part", "none"), full = FALSE)
```

**Arguments**

p	parameter vector
sourcesamp	source marker samples
mixsamp	mixed stock samples
verbose	verbose output?
transf	transformation: "full", "part", or "none"
full	full likelihood (including normalization constants?)
cond	conditional likelihood (not including source sample likelihood?)
debug	debugging output?

**Value**

numeric value of the negative log-likelihood

---

markfreq.condense      *condense marker frequency list*

---

### Description

"condenses" standard marker-frequency-list data (marker samples from sources and mixed stocks) by removing unrepresented markers, lumping markers where that removes no information, etc.

### Usage

```
markfreq.condense(sourcesamp=NULL, mixsamp=NULL, debug=FALSE, exclude.nomix=FALSE)
```

### Arguments

sourcesamp	Turtle data in standard form: list of sourcesamp (see below) and mixsamp OR matrix of source genotype samples (sources=columns, markers=rows)
mixsamp	Vector of mixed stock marker samples
debug	Print out debugging information?
exclude.nomix	Exclude markers not found in the mixed stock?

### Details

Criteria:

- any marker found in the mixed stock but not in any of the sources is deleted from both: it provides no further information about the source contributions (although it does contribute to goodness-of-fit tests)
- any set of markers found in only one source (and possibly also in the mixed stock, although not necessarily) is lumped together
- "exclude.nomix" determines whether to exclude markers not found in the mixed stock

### Value

"Standard" mixed stock analysis data set, suitably condensed.

sourcesamp	source samples
mixsamp	mixed stock samples
err	was there an error in attempting to lump?

### Author(s)

Ben Bolker

### Examples

```
data(lahanas98raw, package="mixstock")
lahan98c <- markfreq.condense(as.mixstock.data(lahanas98raw))
```

---

marknames	<i>Extract/assign names from mixed stock data</i>
-----------	---

---

## Description

Extract names of markers and sources from mixed stock data, or invent names for simulated data

## Usage

```
marknames(x)
sourcenames(x)
mixnames(x)
locnames(x)
mixstock.dimnames(H,R)
label.mixstock.data(x, sourcenames=NULL, marknames=NULL, mixnames=NULL)
```

## Arguments

x	mixstock.data object
H	number of markers
R	number of sources
sourcenames	vector of source names
marknames	vector of marker names
mixnames	vector of mixed stock names

## Details

mixstock.dimnames uses Roman numerals for markers and capital letters for sources. label.mixstock.data assigns names to the data if the source or mixed components of the data set already have names, or use the same rules as mixstock.dimnames. locnames gives a combined vector of source and mixed stock names.

## Value

character vectors of names, or a list with marker and source names

---

mcmc.chainlength.est *Estimate appropriate chain length for mixed stock analysis by MCMC*

---

### Description

Determines an appropriate chain length for MCMC estimation of source contributions to a mixed stock by running Raftery and Lewis and Gelman & Rubin diagnostics repeatedly until convergence criteria are met

### Usage

```
mcmc.chainlength.est(x, mult=1, inflate=sqrt(2), GR.crit=1.2,
nchains=x$R, verbose=FALSE)
```

### Arguments

x	Mixed stock analysis data (a <code>mixstock.data</code> object or a list containing <code>sourcesamp</code> and <code>mixsamp</code> )
mult	How many different times to run tests
inflate	How much to increase chain length at every failed iteration of Gelman and Rubin
GR.crit	Maximum value for Gelman and Rubin 97.5% quantile in order to declare convergence
nchains	number of separate MCMC chains to run
verbose	print lots of detail while running?

### Details

If `mult` is 1, runs Raftery and Lewis diagnostics on a chain starting from equal contributions; if `mult` is greater than 1, runs them on as many chains as there are sources, each starting from a 95% contribution from that source. Iteratively increases each chain length to that suggested by the R&L diagnostic, until all chains pass. Then runs Gelman and Rubin on a set of chains starting from each source. If `mult` is greater than 1, it does each step on `mult` different chains and takes the maximum.

### Value

The maximum chainlength needed to get convergence in all tests

### Author(s)

Ben Bolker

### See Also

`gibbsC`

**Examples**

```
data(simex)
mcmc.chainlength.est(simex, verbose=TRUE)
```

---

mixstock.boot	<i>Bootstrap samples of mixed stock analysis data</i>
---------------	---

---

**Description**

Create a bootstrap (multinomial) sample from a given set of marker data (source and mixed population genotypes), either parametric or non-parametric

**Usage**

```
mixstock.boot(x, param=FALSE, condense=TRUE, save.freq=FALSE,
  param.match="mean")
```

**Arguments**

x	"Standard" mixstock data, a list with <code>mixsamp</code> (vector of samples in the mixed population) and <code>sourcesamp</code> (matrix of samples in the sources, rows=markers, columns=sources)
param	parametric bootstrapping or not?
condense	use <code>markfreq.condense</code> on the results?
save.freq	save frequencies?
param.match	match mean or mode of distribution when parametric bootstrapping?

**Details**

Nonparametric bootstrapping just resamples the observed data from the mixed population and from each source with replacement; this is equivalent to taking a multinomial sample with the probabilities equal to the observed sample frequencies. Parametric bootstrapping takes the observed samples and resamples the probabilities themselves from a Dirichlet distribution, then takes a multinomial sample.

**Value**

A bootstrapped data set, in the same format as the input data: i.e.,

<code>mixsamp</code>	samples in mixed population
<code>sourcesamp</code>	samples in sources

...

**Author(s)**

Ben Bolker

**Examples**

```

true.freq <- matrix(c(0.65,0.33,0.01,0.01,
                    0.33,0.65,0.01,0.01),ncol=2)
true.contrib <- c(0.9,0.1)
x <- simmixstock0(true.freq,true.contrib,50,100,1004)
nboot <- 1000
boot.results.par <- matrix(NA,ncol=12,nrow=nboot)
boot.results.npar <- matrix(NA,ncol=12,nrow=nboot)
for (i in 1:nboot) {
  x.par <- mixstock.boot(x,param=TRUE,condense=FALSE)
  x.npar <- mixstock.boot(x,condense=FALSE)
  boot.results.par[i,] <- c(x.par$sourcesamp,x.par$mixsamp)
  boot.results.npar[i,] <- c(x.npar$sourcesamp,x.npar$mixsamp)
}
summary(boot.results.par[,7:8])
summary(boot.results.npar[,7:8])
par(mfrow=c(1,2))
hist(boot.results.par[,7])
hist(boot.results.npar[,7])

```

---

mixstock.est

*construct (or print) a mixed stock analysis estimate object*


---

**Description**

combine a variety of data into a list with class "mixstock.est"

**Usage**

```

mixstock.est(fit, resample = NULL, data = NULL, em = FALSE, sourcesamp =
NULL, mixsamp = NULL, R = NULL, H = NULL, M=1, transf = "full", method =
"unknown", boot.method = "none", boot.data = NULL, GR = NULL, prior =
NULL)
## S3 method for class 'mixstock.est'
print(x, debug=FALSE, ...)

```

**Arguments**

fit	a fit object (from cml, uml, etc.)
resample	resampling information
data	original data
em	(logical) was estimation done by EM algorithm?
sourcesamp	source marker samples
mixsamp	mixed stock samples
R	number of sources
H	number of markers

M	number of mixed stocks
transf	transformation of parameters
method	estimation method (cml,u1l,mcmc)
boot.method	bootstrap method – nonpar, parametric, or mcmc
boot.data	bootstrap data
GR	Gelman-Rubin diagnostic results
prior	prior strength
x	a <code>mixstock.est</code> object
debug	debug?
...	other arguments

**Value**

an object of type `mixstock.est`

---

mm.wbugs	<i>Run many-to-many model via WinBUGS (or JAGS)</i>
----------	---

---

**Description**

Sets up the many-to-many model and passes it to WinBUGS via R2WinBUGS (or R2jags)

**Usage**

```
mm.wbugs(x, sourcesize,
n.iter=20000, n.burnin=floor(n.iter/2),
n.chains=x$R,
n.thin=max(1, floor(n.chains * (n.iter - n.burnin)/1000)),
files.only = FALSE,
inittype=c("dispersed", "random"), bugs.code=c("TO", "BB"),
returntype=c("mixstock", "coda", "bugs"),
pkg=c("WinBUGS", "JAGS"),
mixprior=1,
which.init,
debug=FALSE,
working.directory,...)
write.TO.bugscode(fn,MIX)
```

**Arguments**

x	a <code>mixstock</code> data object
sourcesize	Relative sizes of sources
n.iter	Total length of each chain
n.burnin	Number of burn-in iterations

n.chains	Number of chains (default, number of sources)
n.thin	thinning rate. Must be a positive integer. Set 'n.thin' > 1 to save memory and computation time if 'n.iter' is large. Default is 'max(1, floor(n.chains * (n.iter - n.burnin) / 1000))' which will only thin if there are at least 2000 simulations.
files.only	(unimplemented) don't run WinBUGS, just produce input files
inittype	"dispersed" or "random" depending on how you want multiple chains to be initialized
bugs.code	"TO" or "BB" depending on whether you want old-style (Toshi Okuyama=TO) or more compact but possibly slower (Ben Bolker=BB) code
mixprior	Dirichlet prior for contributions to mixed stocks. Should be either a vector of length (nmix+1) – one extra for the unknown stock – or a single numeric value which will be replicated
which.init	for "dispersed" start with fewer chains than sources, which sources should be used as the dominant sources in the chains? (default is a random sample without replacement from the list of sources)
debug	run BUGS in debug mode? (i.e. don't exit and go back to R automatically)
...	other arguments to bugs
fn	file name to write BUGS code to
MIX	number of mixed stocks
returntype	return value as a <code>mixstock.est</code> object, a CODA object, or a BUGS object?
pkg	which package to use for back-end calculations
working.directory	working directory for BUGS calculations

**Value**

results of WinBUGS run, as a `mixstock.est` object by default: type varies according to `returntype`. `write.TO.code` produces a BUGS model file.

**Note**

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.

`pkg="JAGS"` is still experimental.

**Author(s)**

Ben Bolker

---

mysum	<i>summaries of MCMC output</i>
-------	---------------------------------

---

**Description**

calculates summary statistics of one or more columns of MCMC output

**Usage**

```
mysum(x, names = NULL)
mysumvec(x, names=NULL)
```

**Arguments**

x	a numeric matrix representing MCMC output
names	vector of column names

**Details**

a slight variant on the standard numeric summary: calculates mean, median, standard deviation, and lower and upper 90 and 95 percent quantiles. `mysum` gives a table (original data as rows, various stats as columns), while `mysumvec` collapses the answer to a vector

**Value**

a numeric vector of results, appropriately named

**Examples**

```
x = matrix(runif(1000),ncol=2,dimnames=list(NULL,c("A","B")))
mysum(x)
mysumvec(x)
```

---

nmark	<i>Set and query mixed stock parameters</i>
-------	---

---

**Description**

Sets or queries the number of markers, mixed stocks, or sources, for a given mixed-stock data set or estimate

**Usage**

```
nmark(object)
nmark(object) <- value
nmix(object)
nmix(object) <- value
nsource(object)
nsource(object) <- value
```

**Arguments**

object	a mixed-stock data object or fit
value	an integer

**Value**

Returns the numeric value or sets the value

**Author(s)**

Ben Bolker

---

normcols	<i>scale columns to frequencies</i>
----------	-------------------------------------

---

**Description**

divides each column of a matrix by its sum, reducing sample values to observed frequencies

**Usage**

```
normcols(z)
```

**Arguments**

z	matrix of sampled numbers (integers)
---	--------------------------------------

**Value**

matrix of frequencies

**Note**

This is just a wrapper for `scale(z, center = FALSE, scale = colSums(z))`

---

numderiv	<i>take the numeric derivative of a function</i>
----------	--

---

**Description**

takes the numeric derivative (gradient) of a function by finite differencing

**Usage**

```
numderiv(p, fn, eps = 1e-05, ...)
```

**Arguments**

p	parameter vector
fn	function
eps	finite-difference increment
...	other arguments to fn

**Value**

numeric vector representing the approximate gradient of the function with respect to the parameters

**See Also**

[numericDeriv](#)

---

p.bayes	<i>Empirical Bayes estimate of prior strength for source marker frequencies</i>
---------	---

---

**Description**

Uses an iterative calculation to estimate an appropriate strength for the flat Dirichlet prior probability for source marker frequencies

**Usage**

```
p.bayes(sourcesamp, bold = 1, cut = 1e-04)
```

**Arguments**

sourcesamp	matrix of source samples
bold	starting value for iterations (??)
cut	cutoff value for iterations

**Details**

See Masuda and Pella's paper

**Value**

prior strength parameter (a)

---

packval	<i>Pack and unpack contribution and marker frequencies</i>
---------	--

---

**Description**

Packs/unpacks source contributions and source marker frequencies into a single vector, after transforming them to an uncorrelated (and possibly unbounded) set of parameters.

**Usage**

```
packval(f, r, transf="part")
packval2(x,R=2,H=3)
unpackval(p,R=2,H=2,x.orig=NULL,transf="full",input.only=FALSE)
unpackval2(p,R=2,H=2,transf="full",x.orig=NULL)
```

**Arguments**

f	Source contributions (numeric vector, all between 0 and 1, sum equals 1)
r	Source marker frequencies: matrix of marker frequencies (row=marker, column=source)
x	a numeric vector containing (elements 1 to R) source contributions and (elements R+1 to (R+R*H)) marker frequencies in sources
p	a packed/transformed parameter vector
transf	Transform to unbounded variables? ("full","part","none") (See <a href="#">p.to.q</a> for description/warnings.)
R	number of sources
H	number of markers
x.orig	original data (for extracting source/marker names)
input.only	return only contribution parameters?

**Value**

packval packs source contributions and marker frequencies specified as a separate vector and a matrix: packval2 packs them (i.e. transforms them) when they have already been run together as a vector. Either produces a numeric vector of the transformed values, with length  $(R-1+R*(H-1))$ . unpackval and unpackval2 invert the operation, producing a list

```
input.freq    source contributions
source.freq   marker frequencies in sources
```

or a vector respectively.

**Author(s)**

Ben Bolker

**See Also**[p.to.q](#)**Examples**

```
data(simex)
sourcefreq <- sweep(simex$sourcesamp,2,apply(simex$sourcesamp,2,sum),"/")
packval(c(0.2,0.8),sourcefreq)
packval(c(0.2,0.8),sourcefreq,transf="full")
```

---

plot.mixstock.est	<i>plot mixed stock analysis estimates</i>
-------------------	--

---

**Description**

plots a mixed stock analysis estimate as a point plot with 95% confidence limites

**Usage**

```
## S3 method for class 'mixstock.est'
plot(x, plot.freqs = FALSE,
     sourcectr=FALSE, contrib.lab = "Estimated source contributions",
     sourcefreq.lab = "Estimated source marker freqs",
     markcolors = rainbow(x$H),
     alength=0.25,
     aunits="inches",
     abbrev,...)
```

**Arguments**

x	mixstock estimate
plot.freqs	plot marker frequency estimates?
sourcectr	plot source-centric estimates?
contrib.lab	label for source contribution plot
sourcefreq.lab	label for marker frequency plot
markcolors	colors corresponding to markers
alength	length of error bar ends in many-to-many plots
aunits	units of error bar end lengths in many-to-many plots
abbrev	abbreviate names on horizontal axis in many-to-many plots? Default if TRUE is 3 characters, but may also be an integer specifying the number of characters
...	other arguments to <a href="#">barplot</a>

**Value**

Produces a plot on the current graphics device.

**Note**

The ... argument can contain a great number of optional arguments to `barplot`: see `barplot` and `xypplot` (in the `lattice` package). Among many others, one can specify (for example) `scale=list(x=list(cex=0.6),y=li` for smaller labels on the horizontal axis and a log scale on the vertical axis; `layout=c(3,5)` to change the number of rows and columns of panels; or `as.table=TRUE` to change the ordering of the panel to top-to-bottom, left-to-right. (One can also use `abbreviate=TRUE` in the scale list, but the `abbrev` argument above seems to be more powerful.)

**Examples**

```
data(simex)
u1 = uml(simex)
plot(u1)
plot(u1,plot.freqs=TRUE)
```

---

pm.wbugs

*Run Pella-Masuda model via WinBUGS*

---

**Description**

Sets up the Pella-Masuda model and passes it to WinBUGS via R2WinBUGS

**Usage**

```
pm.wbugs(x,
  n.iter=20000, n.burnin=floor(n.iter/2),
  n.chains=x$R,
  n.thin=max(1, floor(n.chains * (n.iter - n.burnin)/1000)),
  ...)
```

**Arguments**

x	a mixstock data object
n.iter	Total length of each chain
n.burnin	Number of burn-in iterations
n.chains	Number of chains (default, number of sources)
n.thin	thinning rate. Must be a positive integer. Set 'n.thin' > 1 to save memory and computation time if 'n.iter' is large. Default is 'max(1, floor(n.chains * (n.iter - n.burnin) / 1000))' which will only thin if there are at least 2000 simulations.
...	other arguments to bugs

**Value**

a BUGS object (as returned from R2WinBUGS); can be converted to CODA format using [as.mcmc.bugs](#)

**Note**

[tmcmc](#) is in general be much more convenient and efficient than `pm.wbugs`: `pm.wbugs` is included for completeness and testing of WinBUGS methods.

**Author(s)**

Ben Bolker

**See Also**

[tmcmc](#)

**Examples**

```
data(bolten98)
## Not run:
bolten98.wbugs <- pm.wbugs(bolten98,tot=1000,clearWD=TRUE)
bolten98.wbugs

## End(Not run)
```

---

q.to.p

*Real-to-multifrequency transformation*


---

**Description**

Transform a vector of  $n$  real-valued variables in  $(-\text{Inf}, \text{Inf})$  [or  $(0,1)$ ] to a vector of  $n+1$  variables in  $(0,1)$  that sum to 1, or vice versa.

**Usage**

```
q.to.p(q, transf="full")
p.to.q(p, transf="full")
```

**Arguments**

q	Unconstrained/transformed values: vector of $n$ numeric values in $(-\text{Inf}, \text{Inf})$ [if <code>transf="full"</code> ] or $(0,1)$ [if <code>transf="part"</code> ]
p	Vector of $n+1$ numeric values in $(0,1)$ that sum to 1
transf	(character) "full": use arctan transform to transform $(-\text{Inf}, \text{Inf})$ to $(0,1)$ or vice versa; "part": don't; "none": no transform

**Details**

Essentially, this is a transformation from an unconstrained set of variables to a bounded, constrained set of variables. If `contin` is `TRUE`, an arctan transformation ( $v \leftrightarrow \text{atan}(v)/\pi + 0.5$ ) is used to transform  $(-\text{Inf}, \text{Inf})$  to  $(0, 1)$  or vice versa. In either case, the correlated set of variables (which sum to 1) is transformed to an unconstrained set by taking each variable to be a remainder:  $x[1]=x[1]$ ,  $x[2]=x[2]/(1-x[1])$ , and so forth.

**Value**

Vector of transformed values.

**Note**

This transformation is designed to deal with the problems of bounded optimization and constraints. It actually behaves quite badly because small values are transformed to large negative values, messing up the uniform scaling of the parameters. Now that the bounded optimization of `optim` has improved, `contin="full"` may not be a good idea. It's not clear whether the other transformation (remainders) is better or worse than just optimizing on the first  $(n-1)$  components and assuming that the last frequency equals one minus the sum of the rest.

**Author(s)**

Ben Bolker

**Examples**

```
p.to.q(c(0.3,0.3,0.4))
p.to.q(c(0.3,0.3,0.4),transf="part")
q.to.p(c(-4,3))
q.to.p(c(0,0))
q.to.p(c(0.5,0.5),transf="part")
```

---

rdirichlet

*Dirichlet deviates*

---

**Description**

Produces random deviates for the Dirichlet distribution, the multivariate analogue of the beta distribution.

**Usage**

```
rdirichlet(n, alpha)
```

**Arguments**

`n`                    Number of random deviates to generate.  
`alpha`                Vector of shape parameters.

**Details**

The Dirichlet distribution is a multivariate distribution that describes distributions of frequencies. If a multinomial sample  $n_1, \dots, n_N$  is taken from a population, the estimated distribution of frequencies is  $\text{Dirichlet}(n_1 + 1, \dots, n_N + 1)$ . If  $\{g_i\}$  is a set of Gamma deviates with shape parameters  $\{\alpha_i\}$ , then  $\{g_i\}/\text{sum}(\{g_i\})$  is a Dirichlet deviate.

**Value**

A matrix of random deviates (each in a distinct row) from `rdirichlet`

**Author(s)**

Ben Bolker

**Examples**

```
rdirichlet(5,c(7,4,4))
```

---

runsims

*Run mixed stock simulations*

---

**Description**

Run multiple simulations of a mixed stock systems with specified marker frequencies and source contributions, running multiple estimations (bootstrap samples or MCMC chains) for each simulation

**Usage**

```
runsims(sim.n=10, mc.n=10, totsamp=200, which="all",
true.freq=matrix(c(0.65, 0.31, 0.01, 0.01, 0.01, 0.01,
0.31,0.65,0.01,0.01,0.01,0.01),ncol=2),
true.contrib=c(0.9, 0.1), est="MCMC", verbose=FALSE,
fuzz=0.001, nboot=1000, bootrpt=20, minmarks=3)
```

**Arguments**

<code>sim.n</code>	Number of simulations to run
<code>mc.n</code>	Number of bootstrap/MCMC chains to run for each simulation
<code>totsamp</code>	total sample size (to be distributed half in mixed stock and half, evenly, among sources)
<code>which</code>	which markers to use: "common", "rare", or "all"
<code>true.freq</code>	matrix of true marker frequencies in the sources (column=source, row=marker)
<code>true.contrib</code>	vector of true contributions of sources to the mixed stock

est	estimation method: "MCMC" (Markov Chain Monte Carlo), "cml" (conditional max. likelihood), or "uml" (unconditional max. likelihood)
verbose	produce lots of debugging output?
fuzz	"fuzz" parameters for (e.g.) keeping estimated values away from 0/1
nboot	Number of bootstrap samples/length of MCMC chain
bootrpt	Frequency for reporting on the progress of bootstrap code
minmarks	Minimum number of markers to allow for a simulation

**Value**

Array of the results from all simulations and bootstrap samples

**Examples**

```
## mild kluge to drop unneeded dimensions
runsims(sim.n=1,mc.n=1,nboot=100,est="cml")[, ,]
```

---

simex

*A small sample mixed stock analysis data set*

---

**Description**

This data set is just a small example, generated with four markers (haplotypes) and two sources (rookeries: four markers later condensed with `markfreq.condense` to three), using true marker frequencies  $\{0.65, 0.33, 0.01, 0.1\}$  in rookery A and  $\{0.33, 0.65, 0.01, 0.1\}$  in rookery B; and true contributions of 90% from rookery A and 10% from rookery B. In other words, it was generated with the commands `true.freq <- matrix(c(0.65, 0.33, 0.01, 0.01, 0.33, 0.65, 0.01, 0.01), ncol = 2)` `true.contrib <- c(0.9, 0.1)` `simmixstock0(true.freq, true.contrib, sourcesampsize = 50, mixesampsize = 100, rseed = 1004)`

**Usage**

```
data(simex)
```

**Format**

an object, called `simex`, of class `mixstock.data`

---

`simmixstock0`*Simulate a sample from sources and mixed stock*

---

**Description**

Given information on marker frequencies in sources, the contributions of each source to a mixed stock, and sample sizes, generates multinomial samples of markers from sources and the mixed stock.

**Usage**

```
simmixstock0(sourcefreq, input.freq, sourcesampsize, mixsampsize, rseed=NULL)
```

**Arguments**

<code>sourcefreq</code>	Matrix (markers=rows, source=cols) of true marker frequencies in sources
<code>input.freq</code>	True contributions from each source to the mixed stock
<code>sourcesampsize</code>	Sample size from (each) source
<code>mixsampsize</code>	Sample size from mixed stock
<code>rseed</code>	Random-number seed

**Value**

Multinomial sample.

<code>mixsamp</code>	Sample from mixed stock (vector)
<code>sourcesamp</code>	Sample from sources (matrix, markers=rows, source=cols)
...	

**Author(s)**

Ben Bolker

**Examples**

```
true.freq <- matrix(c(0.65,0.33,0.01,0.01,
                    0.33,0.65,0.01,0.01),ncol=2)
true.contrib <- c(0.9,0.1)
x <- simmixstock0(true.freq,true.contrib,sourcesampsize=50,mixsampsize=100,rseed=1004)
```

---

simmixstock1

*Simulate marker frequencies and distributions in a mixed stock*


---

## Description

Functions for simulating marker frequency distributions and samples in source and mixed populations

## Usage

```
simmixstock1(sampsize = NULL, true.freq = matrix(c(0.65, 0.33, 0.01,
0.01, 0.33, 0.65, 0.01, 0.01), ncol = 2), true.contrib = c(0.9,
0.1), boot = FALSE, param = FALSE, data = NULL, rseed = 1004,
nboot = 1000, chainlen = NULL, ests = c("cmlboot.nonpar",
"cmlboot.par", "umlboot.nonpar", "umlboot.par", "mcmc"),
verbose = FALSE, contrib.only = FALSE)
sim.mark.freq(H,R,g.mark,g.source)
```

## Arguments

sampsize	total sampsize: half from mixed population, (1/(2R)) from each source
true.freq	matrix of marker frequencies in sources
true.contrib	contributions from each source to source population
boot	bootstrap existing data?
param	parametric bootstrap?
data	original data set to bootstrap
nboot	number of bootstrap samples
chainlen	chain length for MCMC
ests	list of estimates to produce (parametric or nonparametric bootstrap for CML or UML estimation, MCMC)
H	number of markers
R	number of sources
g.mark	geometric distribution parameter for marker frequency
g.source	geometric distribution parameter for source contribution
rseed	random number seed
contrib.only	save only source contributions in MCMC chain results?
verbose	verbose output?

## Value

sim.mark.freq just returns an HxR matrix of marker simmixstock1 returns a list with a [genboot](#) result for each type of estimate requested;

**Author(s)**

Ben Bolker

**See Also**[genboot](#), [mysumvec](#)

---

simmixstock2	<i>Simulate multiple mixed stocks</i>
--------------	---------------------------------------

---

**Description**

Simulated multiple mixed stocks

**Usage**

```
simmixstock2(sourcefreq, destmat, sourcesize,  
             sourcesampsize, mixesampsize,  
             nmark, nsource, nmix, rseed = NULL)
```

**Arguments**

sourcefreq	matrix of marker frequencies in the source populations
destmat	matrix of contributions of sources to mixed stocks
sourcesize	vector of relative sizes of sources: default is equal sizes for all sources
sourcesampsize	sizes of samples from source populations
mixsampsize	sizes of samples from mixed populations
nmark	number of distinct markers
nsource	number of source populations
nmix	number of mixed populations
rseed	random number seed

**Details**

If `sourcefreq` and `destmat` are specified, computes expected marker frequencies in mixed stocks and simulates accordingly. If `sourcefreq` and/or `destmat` are missing, they are generated randomly with uniform probabilities.

**Value**Returns an object of type `mixstock.data`**Author(s)**

Ben Bolker

---

sourcesize.wbugs      *Run hierarchical model with via WinBUGS*

---

### Description

Sets up the source-size model and passes it to WinBUGS via R2WinBUGS

### Usage

```
sourcesize.wbugs(x,
  n.iter=20000, n.burnin=floor(n.iter/2),
  n.chains=x$R,
  n.thin=max(1, floor(n.chains * (n.iter - n.burnin)/1000)),
  ...)
```

### Arguments

x	a mixstock data object
n.iter	Total length of each chain
n.burnin	Number of burn-in iterations
n.chains	Number of chains (default, number of sources)
n.thin	thinning rate. Must be a positive integer. Set 'n.thin' > 1 to save memory and computation time if 'n.iter' is large. Default is 'max(1, floor(n.chains * (n.iter - n.burnin) / 1000))' which will only thin if there are at least 2000 simulations.
...	other arguments to bugs

### Value

a BUGS object (as returned from R2WinBUGS); can be converted to CODA format using [as.mcmc.bugs](#)

### Author(s)

Ben Bolker

---

startvec0      *Produce (raw) starting vector of parameters for mixed stock analysis*

---

### Description

Provides raw (untransformed) starting vector of source contribution parameters for mixed stock analysis.

**Usage**

```
startvec0(sourcesamp, mixesamp=NULL, type="equal", sd=1, lmin=0.001)
startvec(sourcesamp, mixesamp=NULL, type="equal",
         marktype="sample", a=1, cond=FALSE, transf="full",
         fuzz=0, sd=1)
lsolve(n, s, tol = 1e-05, warn = FALSE)
```

**Arguments**

sourcesamp	Marker frequencies in sources, or a list with source and mixed samples, or a turtle.data object.
mixesamp	Marker frequencies in mixed stock
type	Various options for setting starting contributions. equal: equal contributions from all sources. random: random multinomial sample with equal multinomial probabilities. rand2: random sample from transformed normal variates with mean 0 and standard deviation sd. A number n between 1 and the number of sources inclusive gives a starting condition with 95% of the contribution from source n and the other 5% evenly split between the other sources. The default is to attempt a solution of the linear equation (sourcesamp*f=mixesamp) and use these values as the starting contributions.
sd	standard deviations for starting type rand2
lmin	When doing linear solutions, the boundary values are {lmin, 1-lmin}.
marktype	method for starting marker frequencies: sample uses the observed sample probabilities; random used observed sample probabilities; weighted does a Bayes-weighted start (a la Masuda and Pella)
cond	Conditional likelihood? (i.e. cond=TRUE gives just the parameters for source contributions, not parameters for source marker frequencies)
a	prior strength parameter
transf	transform ("full", "part", or "none")
fuzz	fuzz parameter for moving parameters away from the boundary
n	source samples (HxR matrix)
s	mixed stock samples (vector)
tol	tolerance for linear fit
warn	warn if numeric problems with solution

**Value**

A parameter vector of the contributions from each of the sources: just the raw source contributions in the case of startvec0, or source contributions and possibly marker frequencies (transformed or untransformed) in the case of startvec.

**Note**

lsolve attempts to get a starting value by solving the linear equation (solve(n, s, tol=tol))

**Author(s)**

Ben Bolker

**Examples**

```
data(simex)
startvec0(simex)
startvec(simex,transf="part")
startvec(simex)
```

tmcmc

*Mixed stock analysis by Markov Chain Monte Carlo***Description**

Runs a Gibbs sampler MCMC starting with 95% contribution from each source, then combines the chains

**Usage**

```
tmcmc(data, n.iter=20000, rseed=1001, n.burnin=floor(n.iter/2),
n.thin=max(1, floor(n.chains * (n.iter - n.burnin)/1000)),
verbose=FALSE, fprior=NULL,
contrib.only=TRUE, rptiter=-1,
outfile=NULL, lang="C",a=NULL,gr=FALSE)
gibbs(sourcesamp, mixsamp, a = 1, startiter, maxiter, startfval = NULL,
n.thin = 1, fprior = NULL, rptiter = -1)
```

**Arguments**

data	Data: a mixstock.data object
n.iter	Total length of each chain
n.burnin	Number of burn-in iterations
n.thin	thinning rate. Must be a positive integer. Set 'n.thin' > 1 to save memory and computation time if 'n.iter' is large. Default is 'max(1, floor(n.chains * (n.iter - n.burnin) / 1000))' which will only thin if there are at least 2000 simulations.
rseed	Random-number seed
verbose	Produce lots of output
fprior	Bayesian prior for source contributions
contrib.only	To save memory, store only information about contributions from each source and not about the estimated marker frequencies in each source
rptiter	How often to issue a progress report. Negative numbers mean no reports
outfile	file to use for output
lang	Run the chain in C or R (for debugging/testing purposes only)?

a	prior strength parameter
gr	calculate Gelman-Rubin convergence statistic?
sourcesamp	matrix of marker samples from sources
mixsamp	vector of marker samples from mixed stock
startiter	starting iteration
maxiter	max. number of iterations
startfval	starting source contributions

### Value

Returns an object of type `mixstock.est`

### Author(s)

Ben Bolker

### References

Masuda and Pella

### Examples

```
data(bolten98)
b98c <- markfreq.condense(as.mixstock.data(bolten98))
t1 <- tmcmc(b98c); t1
```

---

uml

*Mixed stock analysis by unconditional maximum likelihood*

---

### Description

Find the unconditional maximum likelihood estimate (jointly estimating marker frequencies in sources) of the contributions of different sources to a mixed stock, by either a direct-search or an expectation-maximization method

### Usage

```
uml(x, method="direct", optmethod="L-BFGS-B", ...)
uml.ds(x, grad=uml.grad, start.type="lsolve", fuzz=0, bounds=1e-4, ndepfac=1000, method="L-BFGS-B", debug=
transf=c("part", "full", "none"), ...)
uml.em(x, prec=1e-8, prior=1)
```

**Arguments**

x	a list with elements <code>mixsamp</code> (a vector of the sampled markers in the mixed stock) and <code>sourcesamp</code> (a matrix, with markers in rows and sources in columns, of markers in the source samples)
<code>optmethod</code>	to be passed to <code>optim</code>
<code>grad</code>	function giving the gradient of the likelihood
<code>start.type</code>	starting values to use: <code>equal</code> (equal contributions from each source); <code>random</code> (multinomial sample with equal probabilities); <code>rand2</code> (sample out of a transformed normal distribution); a number between 1 and the number of sources; that source starts with 0.95 contribution and the rest start with $0.05/(R-1)$ ; default <code>lsolve</code> , the linear solution to the problem
<code>fuzz</code>	min. value (1-min is the max.) for starting contributions
<code>bounds</code>	( <code>bounds</code> , <code>1-bounds</code> ) are the lower and upper bounds for mle calculations
<code>ndepfac</code>	factor for computing numerical derivatives; numerical derivative stepsize is computed as <code>bounds/ndepfac</code> [OBSOLETE with gradient function?]
<code>method</code>	optimization method, to be passed to <code>optim</code>
<code>transf</code>	transformation
<code>debug</code>	produce debugging output?
<code>control</code>	other control arguments to <code>optim</code>
...	other arguments to <code>mle</code> or <code>optim</code> (e.g. <code>hessian=FALSE</code> to suppress (slow) hessian calculation, etc.)
<code>prec</code>	precision for determining convergence of EM algorithm
<code>prior</code>	prior for EM algorithm

**Details**

uml uses either a direct-search algorithm or an EM algorithm to find the ML estimate

**Value**

	an object of class <code>mixstock.est</code> , with elements
<code>fit</code>	information on the ML fit
<code>resample</code>	bootstrap information, if any
<code>data</code>	original data used for estimate
<code>R</code>	number of sources
<code>H</code>	number of markers
<code>contin</code>	estimation done on transformed proportions?
<code>method</code>	optimization method
<code>boot.method</code>	resampling method
<code>boot.data</code>	raw resampling information
<code>gandr.diag</code>	Gelman-Rubin diagnostic information for MCMC estimates
<code>prior</code>	Prior for MCMC estimates
<code>em</code>	estimation done by EM algorithm?

**Author(s)**

Ben Bolker

**Examples**

```

true.freq <- matrix(c(0.65,0.33,0.01,0.01,
                    0.33,0.65,0.01,0.01),ncol=2)
true.contrib <- c(0.9,0.1)
x <- simmixstock0(true.freq,true.contrib,50,100,1004)
uml.est <- uml(x)
uml.est
uml.emest <- uml.em(x)
uml.emest

```

uml.lik

*Likelihoods and gradients of stock analysis models***Description**

Calculate negative log likelihoods and gradients for unconditional and conditional models

**Usage**

```

uml.lik(p, data, transf=c("full","part","none"), verbose=FALSE,
debug=FALSE)
cml.lik(p, sourcefreq, data, transf=c("full","part","none"),
        verbose=FALSE, fulllik=TRUE, debug=FALSE)
cml.grad(p, sourcefreq, data, transf="full",
        verbose=FALSE, fulllik=NULL, debug=FALSE)
uml.grad(p, data, transf="full", debug=FALSE, verbose=FALSE)
dcmat.a(x, debug=FALSE)

```

**Arguments**

p	a vector of parameters.
data	a data set in <code>mixstock.data</code> format
sourcefreq	source frequencies
transf	how are parameters transformed?
verbose	print messages?
debug	debug?
x	vector of parameters
fulllik	for CML, give likelihood corresponding to source samples (test only)?

**Details**

The log likelihood is the log multinomial likelihood of the mixed population samples (`data$mixsamp`) given the expected frequencies in the mixed population, which are computed from the contributions and the source marker frequencies, plus the log multinomial likelihoods of the samples in each source given the marker frequencies specified for each source. `dcmat.a` is a utility function for the gradient calculations.

**Value**

Negative log likelihood, possibly plus a constant corresponding to the normalization factor

**Author(s)**

Ben Bolker

**Examples**

```
data(simex)
rfreq <- normcols(simex$sourcesamp)
tmpf <- function(p) {
  uml.lik(c(p,1-p,rfreq),simex,transf="none")
}
pvec <- seq(0.01,0.99,by=0.01)
plot(pvec,sapply(pvec,tmpf))
```

# Index

- \*Topic **aplot**
  - addlabels.barplot, 2
- \*Topic **array**
  - blockdiag, 7
- \*Topic **datasets**
  - bolten98, 8
  - lahanas98, 20
  - simex, 38
- \*Topic **distribution**
  - rdirichlet, 36
- \*Topic **misc**
  - AIC.mixstock.est, 3
  - as.mcmc.bugs, 4
  - as.mixstock.data, 5
  - as.mixstock.est.bugs, 6
  - BUGS.out, 9
  - calc.GR, 9
  - calc.mult.GR, 10
  - calc.RL.0, 11
  - cml, 13
  - coef.toptim, 14
  - expand.bugs.data, 15
  - genboot, 15
  - get.bot, 17
  - gibbsC, 18
  - gibbsmat, 19
  - hwdiag.check, 20
  - loglik2.C, 21
  - markfreq.condense, 22
  - marknames, 23
  - mcmc.chainlength.est, 24
  - mixstock.boot, 25
  - mixstock.est, 26
  - mm.wbugs, 27
  - mysum, 29
  - nmark, 29
  - normcols, 30
  - numderiv, 31
  - p.bayes, 31
  - packval, 32
  - plot.mixstock.est, 33
  - pm.wbugs, 34
  - q.to.p, 35
  - runsims, 37
  - simmixstock0, 39
  - simmixstock1, 40
  - simmixstock2, 41
  - sourcesize.wbugs, 42
  - startvec0, 42
  - tmcmc, 44
  - uml, 45
  - uml.lik, 47
- addlabels.barplot, 2
- AIC.mixstock.est, 3
- as.mcmc.bugs, 4, 35, 42
- as.mixstock.data, 5
- as.mixstock.est (AIC.mixstock.est), 3
- as.mixstock.est.bugs, 6
- as.mixstock.est.mcmc  
(as.mixstock.est.bugs), 6
- barplot, 6, 33, 34
- blockdiag, 7
- bolten98, 8
- bolten98raw (bolten98), 8
- boxplot.mixstock.est  
(AIC.mixstock.est), 3
- BUGS.out, 9
- calc.GR, 9
- calc.mult.GR, 10
- calc.mult.RL (calc.RL.0), 11
- calc.RL.0, 11
- cml, 13, 16
- cml.grad (uml.lik), 47
- cml.lik (uml.lik), 47
- coef.mixstock.est (AIC.mixstock.est), 3
- coef.toptim, 14

- confint.mixstock.est  
    (AIC.mixstock.est), 3
- dcmat.a (uml.lik), 47
- do.call, 7
- expand.bugs.data, 15
- genboot, 15, 40, 41
- get.bot, 17
- get.bse (get.bot), 17
- get.ct1 (get.bot), 17
- get.frq (get.bot), 17
- get.mix (get.bot), 17
- get.mp.input (get.bot), 17
- gibbs (tmcmc), 44
- gibbsC, 9, 18
- gibbsmat, 19
- gibbsrpt (gibbsmat), 19
- hist.mixstock.est (AIC.mixstock.est), 3
- hwdiag.check, 20
- label.mixstock.data (marknames), 23
- lahanas98, 20
- lahanas98raw (lahanas98), 20
- locnames (marknames), 23
- logLik.mixstock.est (AIC.mixstock.est),  
    3
- logLik.toptim (coef.toptim), 14
- loglik2.C, 21
- lsolve (startvec0), 42
- markfreq.condense, 22
- marknames, 23
- mcmc.chainlength.est, 24
- mixnames (marknames), 23
- mixstock.boot, 25
- mixstock.data, 47
- mixstock.data (as.mixstock.data), 5
- mixstock.dimnames (marknames), 23
- mixstock.est, 14, 26
- mm.wbugs, 27
- mysum, 29
- mysumvec, 41
- mysumvec (mysum), 29
- nmark, 29
- nmark<- (nmark), 29
- nmix (nmark), 29
- nmix<- (nmark), 29
- normcols, 30
- nsource (nmark), 29
- nsource<- (nmark), 29
- numderiv, 31
- numericDeriv, 31
- optim, 13, 46
- p.bayes, 31
- p.to.q, 32, 33
- p.to.q (q.to.p), 35
- packval, 32
- packval2 (packval), 32
- plot.mixstock.data (as.mixstock.data), 5
- plot.mixstock.est, 33
- plotvar (gibbsmat), 19
- pm.wbugs, 34
- print.mixstock.data (as.mixstock.data),  
    5
- print.mixstock.est (mixstock.est), 26
- put.bse (get.bot), 17
- put.ct1 (get.bot), 17
- put.mix (get.bot), 17
- put.mp (get.bot), 17
- q.to.p, 35
- rdirichlet, 36
- RL.burn (calc.RL.0), 11
- RL.max (calc.RL.0), 11
- runsims, 37
- sim.mark.freq (simmixstock1), 40
- simex, 38
- simmixstock0, 39
- simmixstock1, 40
- simmixstock2, 41
- sourcenames (marknames), 23
- sourcesize.wbugs, 42
- startvec, 16
- startvec (startvec0), 42
- startvec0, 16, 42
- summary.mixstock.est  
    (AIC.mixstock.est), 3
- tmcmc, 9, 35, 44
- uml, 45
- uml.grad (uml.lik), 47

uml.lik, [47](#)  
unpackval (packval), [32](#)  
unpackval2 (packval), [32](#)  
write.TO.bugscode (mm.wbugs), [27](#)