Practical: Using simmr

Andrew Parnell

Introduction

Welcome to the practical! We will learn about:

- Loading data into simmr
- Running simmr
- Getting output from simmr
- Using simmr for single observations
- Setting prior distributions on the dietary proportions
- Customising your own simmr output

It's assumed that you already have `simmr` installed via:

```
library(devtools)
install_github('andrewcparnell/simmr')
```

You should then be able to run

```
library(simmr)
```

without error.

This document is a slightly friendlier version of the simmr vignette.

You should follow and run the commands shown in the grey boxes below. At various points you will see a horizontal line in the text which indicates a question you should try to answer, like this:

What words does the following command print to the console?

```
print("Hello World")
```

If you get stuck, please get our attention and we will try to help! There are no prepared answers to these questions so keep you own record as you go. At the end of the practical are harder questions which you can attempt if you get through all of the material. If you find any mistakes in the document please let us know.

You can run the code from these practicals by loading up the `.Rmd` file in the same directory in Rstudio. This is an R markdown document containing all the text. Feel free to add in your own answers, or edit the text to give yourself extra notes. You can also run the code directly by highlighting the relevant code and clicking `Run`.

Loading data into simmr

Unlike SIAR and MixSIAR which have both menu systems and script-based ways of interacting with the software, simmr only allows a script-based version. This makes it slightly harder to use for those unfamiliar with R but allows for much more flexibility in the long run. We will focus on using the script-based way of running simmr.

The key differences between simmr and other SIMMs are:

- simmr has a slightly richer mixing model than SIAR, based on code from the Parnell et al 2013 Environmetrics paper
- simmr uses ggplot2 to create graphs and JAGS to run the mixing model
- simmr has simple functions to load data, create iso-space plots, and summarise/plot output
- simmr has some simple functions for comparing and combining sources, and for comparing groups

We will use the data included with SIAR (make sure you have installed SIAR using the instructions in the 'using SIAR' practical) to run simmr. When running your own models you should try to keep your data in the same format as these examples. At minimum you need two files to get simmr working; a consumers file and a sources file. The simplest geese data set is obtained via:

```
library(siar)
data(geese1demo)
print(geese1demo)
```

This data set has two columns, one for each isotope, and 9 individuals. A useful command for learning about the structure of an R data set is `str`, especially for large data objects:

```
str(geese1demo)
```

```
##  num [1:9, 1:2] 10.2 10.4 10.4 10.5 10.2 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:2] "d15NPl" "d13CPl"
```

We can see that it has 9 rows, 2 columns, and is of numeric (`num`) mode. The two column labels refer to the δ^{15}N and δ^{13}C isotope values.

The sources data can be obtained from:

```
data(sourcesdemo)
print(sourcesdemo)
```

```
##        Sources  Meand15N    SDd15N  Meand13C    SDd13C
## 1      Zostera  6.488984 1.4594632 -11.17023 1.2149562
## 2        Grass  4.432160 2.2680709 -30.87984 0.6413182
## 3    U.lactuca 11.192613 1.1124385 -11.17090 1.9593306
## 4 Enteromorpha  9.816280 0.8271039 -14.05701 1.1724677
```

We can see that there are 4 sources, with their names in the first column. The remaining columns refer to the means and standard deviations for each source on each isotope. The isotopes need to be in the same order as the consumer data in `geese1demo`. Note the structure of this object

```
str(sourcesdemo)
```

```
## 'data.frame':    4 obs. of  5 variables:
##  $ Sources : Factor w/ 4 levels "Enteromorpha",..: 4 2 3 1
##  $ Meand15N: num  6.49 4.43 11.19 9.82
##  $ SDd15N  : num  1.459 2.268 1.112 0.827
##  $ Meand13C: num  -11.2 -30.9 -11.2 -14.1
##  $ SDd13C  : num  1.215 0.641 1.959 1.172
```

It's a data frame. This is an R data type which can store both text and numbers, useful for storing the source names as well as their isotope values.

We could run simmr with just these two data files. However, this would produce a pretty poor model as we don't have any corrections for trophic enrichment factors (TEFs). The TEFs file looks just like the source file:

```r
data(correctionsdemo)
print(correctionsdemo)
```

```
##          Source Mean15N Sd15N Mean13C Sd13C
## 1       Zostera    3.54  0.74    1.63  0.63
## 2         Grass    3.54  0.74    1.63  0.63
## 3     U.lactuca    3.54  0.74    1.63  0.63
## 4 Enteromorpha    3.54  0.74    1.63  0.63
```

Lastly, if we have concentration dependence we can include these too. The data for the Geese comes from:

```r
data(concdepdemo)
print(concdepdemo)
```

```
##         Sources Meand15N SDd15N Meand13C SDd13C
## 1       Zostera   0.0297 0.0097   0.3593 0.0561
## 2         Grass   0.0355 0.0063   0.4026 0.0380
## 3     U.lactuca   0.0192 0.0053   0.2098 0.0327
## 4 Enteromorpha   0.0139 0.0057   0.1844 0.1131
```

Note that although this data set includes standard deviations on the concentration dependencies, they are currently not used by simmr in the model run.

If you were loading these data sets in yourself, it's best to store them in the same directory and then load them in from there, e.g.:

```r
# Set the working directory (where R looks first for files)
setwd('path/to/files')
# Read in consumers
consumers = read.table('my_consumer_file.txt',header=TRUE)
# Read in sources
sources = read.table('my_sources_file.txt',header=TRUE)
# Read in TEFs
TEFs = read.table('my_TEF_file.txt',header=TRUE)
```

The extra `header=TRUE` argument tells R that there are column names at the top of the file.

---

1. What is the structure of the TEFs object? How many rows and columns does it have?
2. There's another data object that comes with siar called `geese2demo`. How many rows and columns does this have?
3. Create some simple scatter plots of the `geese1demo` data using `plot`. See if you can add in the source means corrected for the TEF means (hint: add the means together and then plot using `points`)

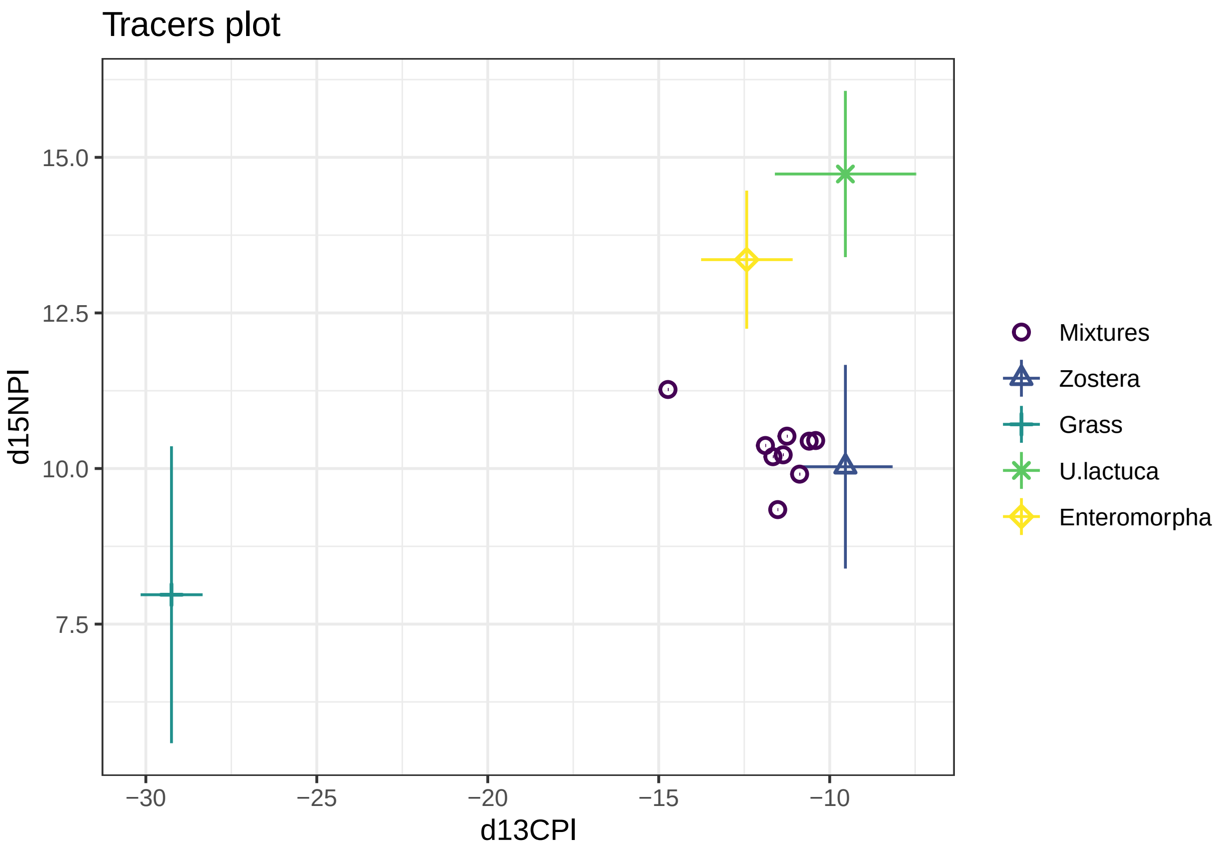
Loading data into simmr

We are now in a position to run simmr. The first function to use is `simmr_load`. You can find help on this function by typing the name with a `?` in front. If you are using Rstudio you can use the `<TAB>` key to complete your command once you have typed in the first few characters. To load the Geese data into simmr, type:

```r
simmr_in = simmr_load(mixtures = geese1demo[,c(2,1)],
                      source_names = as.character(sourcesdemo[,1]),
                      source_means = sourcesdemo[,c(4,2)],
                      source_sds = sourcesdemo[,c(5,3)],
                      correction_means = correctionsdemo[,c(4,2)],
                      correction_sds = correctionsdemo[,c(5,3)],
                      concentration_means = concdepdemo[,c(4,2)])
```

Once loaded we can create an isospace plot very simply with:

```
plot(simmr_in)
```



You should see that the consumers are inside the mixing polygon (or *convex hull*) of the sources. The consumers are close to the Zostera source, so we would expect this to come out as the main dietary proportion.

We can also print out the details of the current model:

```
print(simmr_in)
```

```
## This is a valid simmr input object with 9 observations, 2 tracers, and 4 sources.
## The source names are: Zostera, Grass, U.lactuca, Enteromorpha.
## The tracer names are: d13CPl, d15NPl.
```

Running simmr

To run simmr we use the `simmr_mcmc` function:

```
simmr_out = simmr_mcmc(simmr_in)
```

simmr now runs the MCMC algorithm (using JAGS) and, whilst running, reports the percentage complete. When finished, the first thing to do is to check convergence

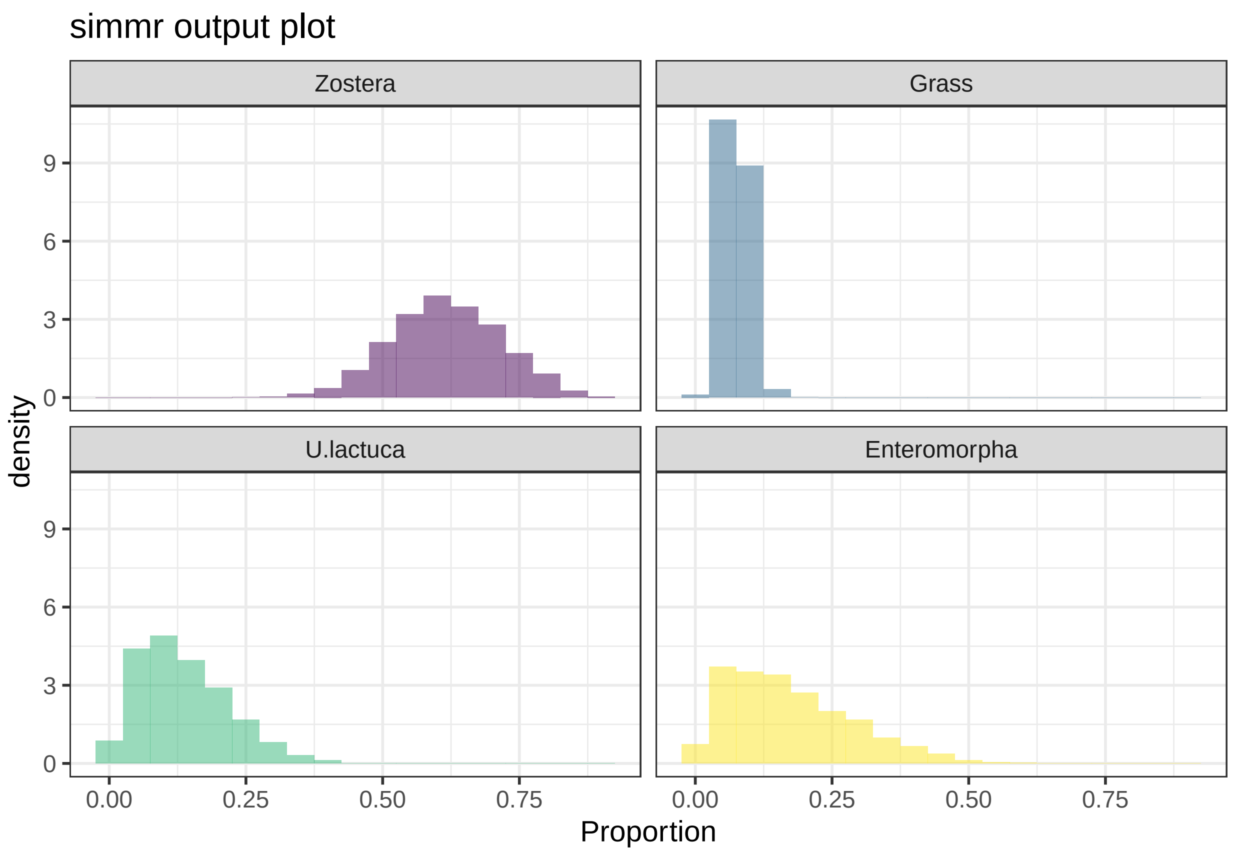
```
summary(simmr_out, type = 'diagnostics')
```

```
##
## Summary for 1
## Gelman diagnostics - these values should all be close to 1.
## If not, try a longer run of simmr_mcmc.
##      deviance      Zostera        Grass    U.lactuca Enteromorpha   sd[d13CPl]
##             1            1            1            1            1            1
##    sd[d15NPl]
##             1
```

These are the Brooks-Gelman-Rubin diagnostics discussed in the lecture notes. All values are close to 1 so the model has achieved satisfactory convergence. The rule of thumb is usually that the model has converged ok if the values are less than 1.1.

We can plot the posterior distributions of the dietary proportions with:

```
plot(simmr_out, type = 'histogram')
```



If you want more textual output you can get it with:

```
summary(simmr_out, type = c('statistics', 'quantiles'))
```

```
##
## Summary for 1
##                2.5%    25%    50%    75%  97.5%
## deviance     51.206 52.652 54.212 56.448 62.905
## Zostera       0.423  0.544  0.613  0.686  0.811
## Grass         0.035  0.059  0.073  0.088  0.120
## U.lactuca     0.020  0.072  0.124  0.187  0.322
## Enteromorpha  0.021  0.082  0.155  0.247  0.434
## sd[d13CPl]    0.045  0.514  0.877  1.298  2.342
## sd[d15NPl]    0.020  0.176  0.374  0.650  1.390
##                 mean    sd
## deviance      54.948 3.115
## Zostera        0.616 0.101
## Grass          0.074 0.022
## U.lactuca      0.136 0.082
## Enteromorpha   0.174 0.113
## sd[d13CPl]     0.952 0.614
## sd[d15NPl]     0.465 0.395
```

This will produce a quantiles for the posterior proportions (useful for 50% and 95% uncertainty intervals), and the estimated mean and standard deviation It will also give the same estimates for the residual standard deviations for each isotope. These will tend to be large when the consumers lie outside the source mixing polygon.

If you want held on these functions use `?summary.simmr_output` and `?plot.simmr_output`

---

1. What's the structure of the `simmr_out` object? Can you see anything you recognise in it? Try accessing different parts of it using the `$` notation, e.g. `simmr_out$input$n_obs`
2. Try the command `plot(simmr_out, type = 'matrix')`. What does this produce? What other types of plots are available?
3. Try running the model again without including the `correctionsdemo` data. What happens to the isospace plot?

Longer simmr runs

If you want to be really certain of convergence you can run simmr for more iterations with some extra arguments. The extra arguments are:

1. `iter` which sets the total number of iterations. The default is 10,000
2. `burn` which sets the number of initial iterations to remove. The default is 1,000
3. `thin` which sets the amount of thinning (removal) of iterations to avoid autocorrelation in the output values. The default is 10, which means simmr will keep only every 10th iteration
4. `n.chain` which sets the number of chains (i.e. the number of runs with different starting values). The default is 4 chains.

Usually the default values will be fine, but you could double them if you wanted a longer run. The resulting number of iterations kept by simmr for the posterior distribution is `n.chain*(iterations-burnin)/thinby`. It's usually not a good idea to store more than 10,000 iterations unless you have lots of RAM.

To include such extra values a run might be:

```
simmr_out_long = simmr_mcmc(simmr_in,
                           mcmc_control = list(iter = 20000,
                                               burn = 2000,
                                               thin = 20,
                                               n.chain = 4))
```

---

1. Without checking, how many iterations will the command above save?
2. Did the results change much between the shorter and longer run?
3. Were the convergence results better for the longer run (i.e. were the BGR diagnostic values closer to 1)?

Working with multiple groups

Sometimes you might be interested in running simmr for multiple different groups of consumers. These different groups might be different sexes, different sampling periods, different locations, etc. simmr will run these simultaneously and store the output for easier plots and comparison.

The data which are included in simmr for multiple groups analysis can be found with:

```r
data(geese2demo)
head(geese2demo,15)
```

```
##       Group d15NPl d13CPl
##  [1,]     1  10.22 -11.36
##  [2,]     1  10.37 -11.88
##  [3,]     1  10.44 -10.60
##  [4,]     1  10.52 -11.25
##  [5,]     1  10.19 -11.66
##  [6,]     1  10.45 -10.41
##  [7,]     1   9.91 -10.88
##  [8,]     1  11.27 -14.73
##  [9,]     1   9.34 -11.52
## [10,]     2  11.68 -15.89
## [11,]     2  12.29 -14.79
## [12,]     2  11.04 -17.64
## [13,]     2  11.46 -16.97
## [14,]     2  11.73 -17.25
## [15,]     2  12.29 -14.77
```

```r
str(geese2demo)
```

```
##  num [1:251, 1:3] 1 1 1 1 1 1 1 1 1 2 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:3] "Group" "d15NPl" "d13CPl"
```

This is a much bigger data set. The first column contains the group number. To run simmr you now need to specify the groups so that it can separate out the model. We can see how many and how large the groups are with:

```r
table(geese2demo[,'Group'])
```

```
##
##  1  2  3  4  5  6  7  8
##  9 29 74 10 41 20 32 36
```

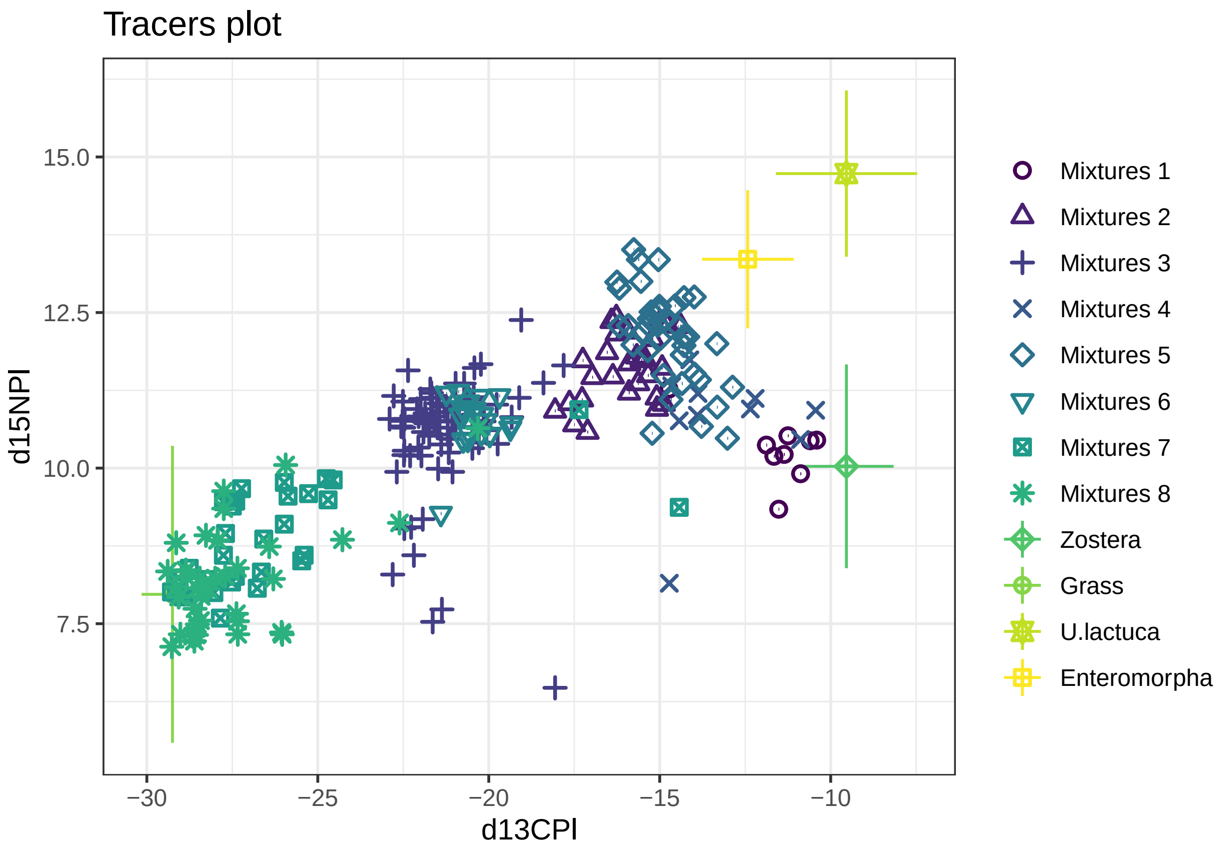
so 8 groups ranging from 9 to 74 observations. simmr will work with up to 30 groups. There needs to be at least 3 observations per group for simmr to run, but really 5 or more is desirable if you want to properly estimate the residual error.

To run simmr with this data set it's the same as before:

```r
simmr_in2 = simmr_load(mixtures = geese2demo[,c(3,2)],
                       source_names = as.character(sourcesdemo[,1]),
                       source_means = sourcesdemo[,c(4,2)],
                       source_sds = sourcesdemo[,c(5,3)],
                       correction_means = correctionsdemo[,c(4,2)],
                       correction_sds = correctionsdemo[,c(5,3)],
                       concentration_means = concdepdemo[,c(4,2)],
                       group = as.integer(geese2demo[,1]))
```

Again we can do an isospace plot with:

```
plot(simmr_in2, group = 1:8)
```



We run simmr the same as before too:

```
simmr_out2 = simmr_mcmc(simmr_in2)
```

The summaries and convergence diagnostics are created with, e.g.:

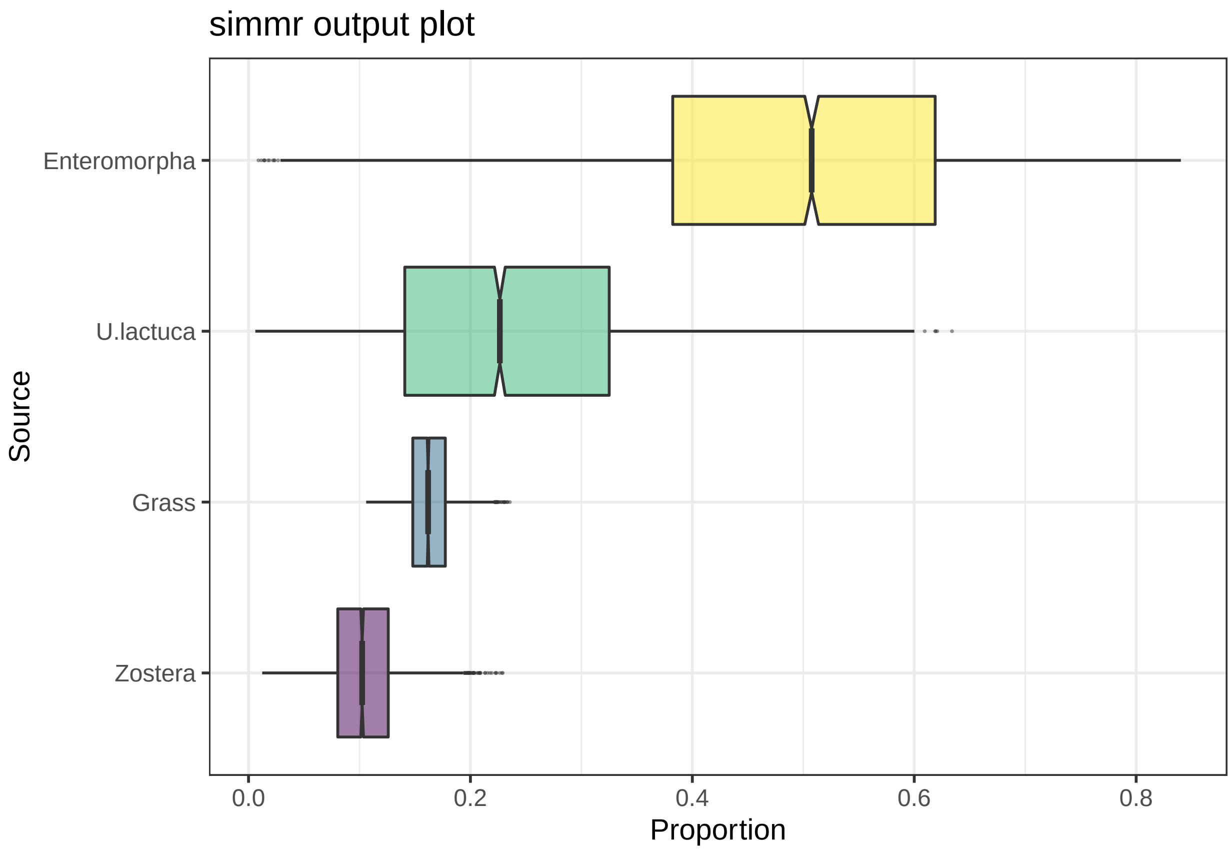
```
summary(simmr_out2, type = c('diagnostics', 'quantiles'), group = 3)
```

```
##
## Summary for 3
## Gelman diagnostics - these values should all be close to 1.
## If not, try a longer run of simmr_mcmc.
##     deviance      Zostera         Grass     U.lactuca Enteromorpha    sd[d13CPl]
##            1            1             1             1            1             1
##   sd[d15NPl]
##            1
##                 2.5%    25%    50%    75%  97.5%
## deviance      51.234 52.570 54.117 56.334 63.175
## Zostera        0.414  0.551  0.616  0.685  0.809
## Grass          0.035  0.060  0.074  0.088  0.119
## U.lactuca      0.020  0.069  0.121  0.187  0.329
## Enteromorpha  0.022  0.082  0.149  0.243  0.442
## sd[d13CPl]     0.054  0.475  0.843  1.270  2.451
## sd[d15NPl]     0.015  0.170  0.357  0.641  1.425
```

Running the above without the `group` argument gives the results for all groups (though is pretty long).

We can get some within-group boxplots with:

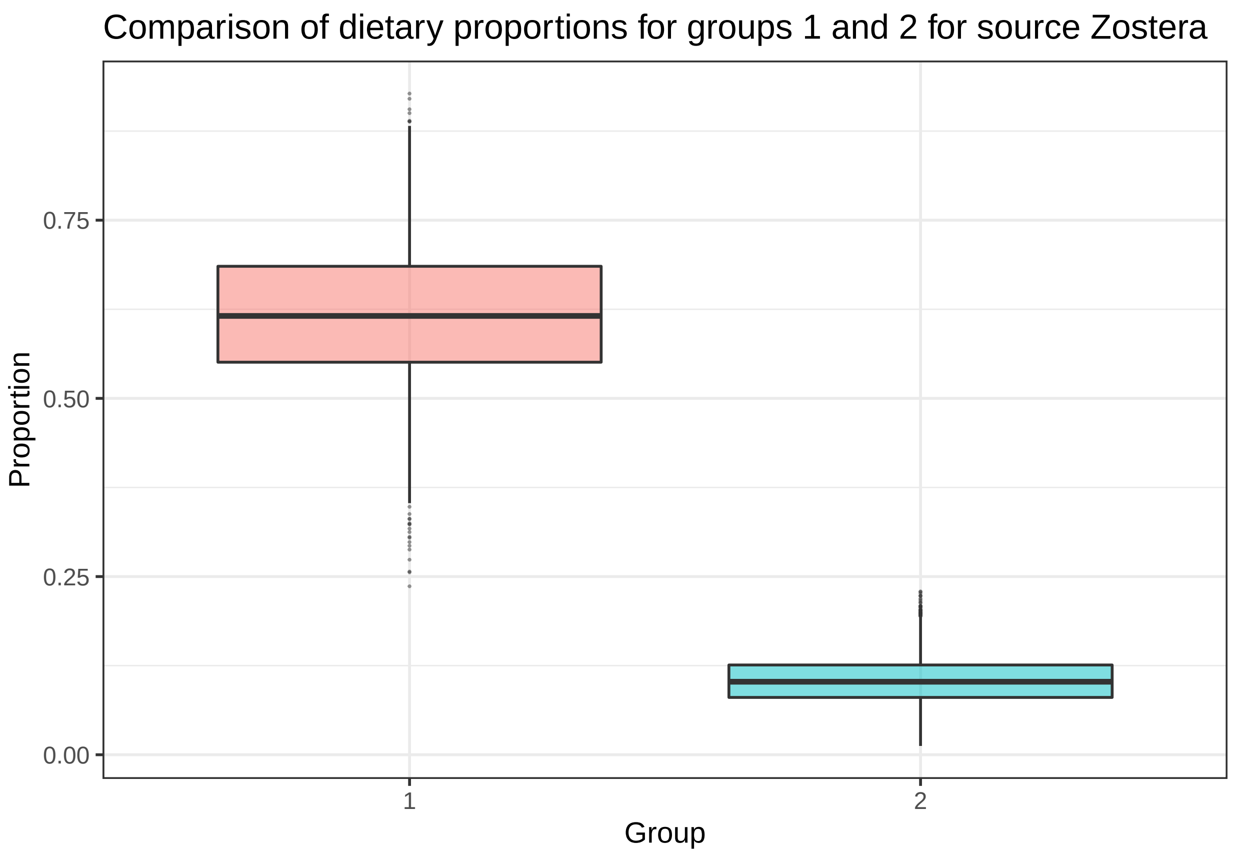
```r
plot(simmr_out2, type = 'boxplot', group = 2)
```



To compare groups we can run, e.g.:

```r
compare_groups(simmr_out2,source='Zostera',groups=1:2)
```

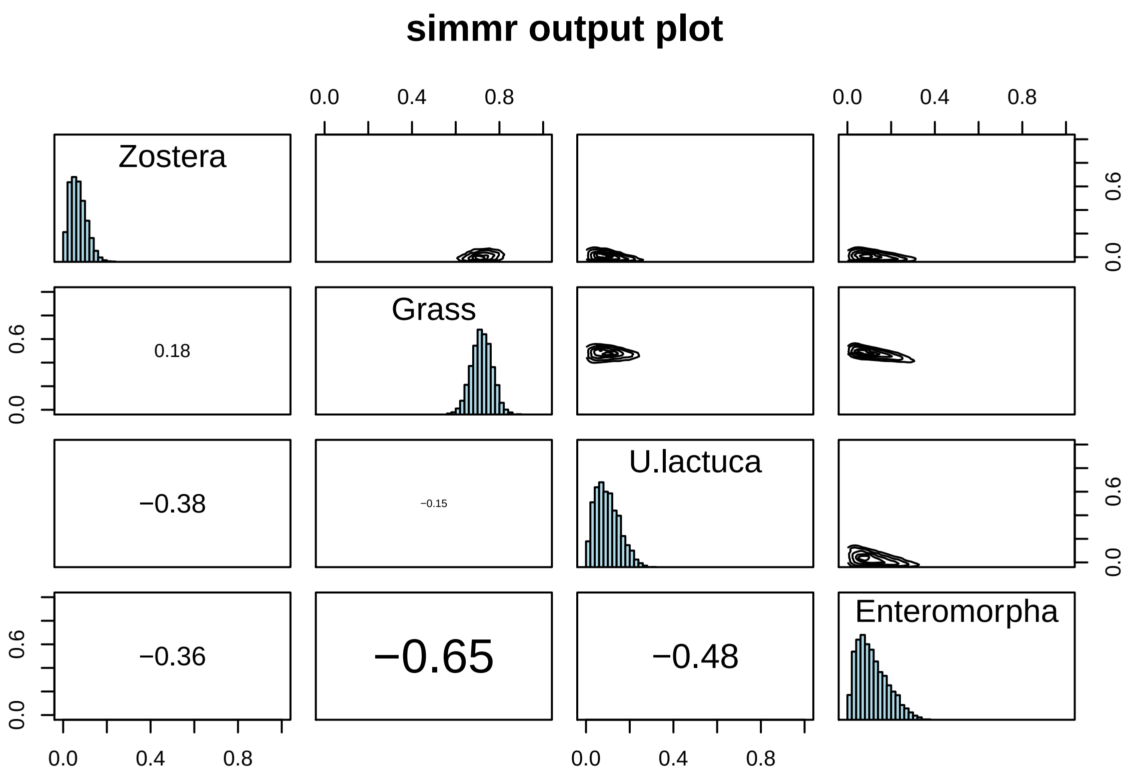
```
## Prob ( proportion of Zostera in group 1 > proportion of Zostera in group 2 ) = 1
```



This will also estimate the probability that this source is being eaten by one group more than the other.

Finally, the matrix plot (discussed in the lectures) can be created with:

```
plot(simmr_out2, type = 'matrix', group = 7)
```



This is a really useful plot as it provides the histograms and the relationships between the sources, potentially identifying which sources are impossible to discern between in the model. It takes a little bit of practice to interpret a matrix plot.

Running the model for individual observations

When you have just a single observation it is impossible to estimate the residual standard deviation. However you can still estimate the dietary proportions and simmr will run this as standard when provided with only one observation by forcing the prior distribution on the residual standard deviation to be tiny. We can create a simple example by just taking a single row from the geese data:

```
simmr_in3 = simmr_load(mixtures = geese1demo[1,c(2,1), drop = FALSE],
                       source_names = as.character(sourcesdemo[,1]),
                       source_means = sourcesdemo[,c(4,2)],
                       source_sds = sourcesdemo[,c(5,3)],
                       correction_means = correctionsdemo[,c(4,2)],
                       correction_sds = correctionsdemo[,c(5,3)],
                       concentration_means = concdepdemo[,c(4,2)])
simmr_out3 = simmr_mcmc(simmr_in3)
```

The extra argument `drop = FALSE` is used above because the mixtures have to be in matrix format (without this R will turn the resulting object into a vector rather than a matrix).

```
summary(simmr_out3, type = 'quantiles')
```

```
##
## Summary for 1
##                2.5%   25%   50%   75%  97.5%
## deviance      4.222 4.517 4.929 5.770 10.016
## Zostera       0.257 0.452 0.571 0.682  0.850
## Grass         0.018 0.045 0.067 0.090  0.138
## U.lactuca    0.019 0.080 0.148 0.251  0.486
## Enteromorpha 0.020 0.077 0.147 0.265  0.523
## sd[d13CPl]   0.000 0.000 0.001 0.001  0.001
## sd[d15NPl]   0.000 0.000 0.001 0.001  0.001
```

Note that the estimates of the residual standard deviation are all tiny values.

Adding in your own prior information

Occasionally it is the case that previous studies have given insight into the likely values of the dietary proportions for your study. You can use this external information to guide the model by changing the prior distributions used for the transformed CLR-distribution (see module on: 'From SIAR to MixSIAR'). If prior information is available, it is usually a good idea to use it, as it means the model will often converge quicker, and yield more realistic results.

simmr has a special function for the inclusion of prior distributions. For example suppose we had prior information that the four sources in the Geese example had means (0.5, 0.1, 0.2, 0.2) and standard deviations (0.1, 0.05, 0.01, 0.02), we would run:

```
prior = simmr_elicit(n_sources = 4,
                     proportion_means = c(0.5,0.1,0.2,0.2),
                     proportion_sds = c(0.1,0.05,0.01,0.02))
```

This requires some optimisation routines in the background and can often be slow. The function will print out some values to use in the prior distribution for the CLR-transformed values. We can use these as follows:

```
simmr_out_prior = simmr_mcmc(simmr_in,
                             prior_control = list(means = prior$mean,
                                                  sd = prior$sd))
```

---

1. Try several different prior estimates and see how they change the posterior dietary proportions with `plot`.
2. An alternative default prior for simmr would be when all of the mean prior proportions are set to 1 divided by the number of sources, and the prior standard deviations are set to be very large. What effect does using this prior have on the different Geese data sets?

Creating your own plots and tables

Often what you want to create isn't exactly part of the simmr toolkit. Maybe the plots don't look right, or maybe you want to compare two different groups in a particular way. To do this, you can get at the simmr output yourself, and then play with it as you want.

Whenever simmr creates the dietary proportions using e.g. `simmr_mcmc`, it stores the output as an R *list*. You can see everything in the list with:

```
str(simmr_out)
```

This will provide quite a lot of output, but the most important part is the element named `output` which contains all of the posterior samples. You can see the first few samples with:

```
head(simmr_out$output[[1]]$BUGSoutput$sims.matrix)
```

```
##      deviance   Zostera      Grass  U.lactuca Enteromorpha sd[d13CPl]
## [1,] 53.53697 0.5872722 0.05757218 0.19916548  0.155990187  0.8953026
## [2,] 54.93214 0.7991078 0.08078688 0.05489919  0.065206114  0.2820967
## [3,] 51.54679 0.6077575 0.07016951 0.12372173  0.198351208  0.9406384
## [4,] 57.88267 0.7573788 0.11415508 0.12434936  0.004116792  1.6248838
## [5,] 58.23980 0.5363323 0.06614164 0.36843126  0.029094802  1.6951408
## [6,] 53.46256 0.6866064 0.05790994 0.02437947  0.231104142  1.1590801
##      sd[d15NPl]
## [1,]  0.5813515
## [2,]  0.8195730
## [3,]  0.3516759
## [4,]  0.7456589
## [5,]  0.2813865
## [6,]  0.4202864
```

or you can access individual parameters with:

```
head(simmr_out$output[[1]]$BUGSoutput$sims.list$p)
```

```
##        Zostera      Grass  U.lactuca Enteromorpha
## [1,] 0.5872722 0.05757218 0.19916548  0.155990187
## [2,] 0.7991078 0.08078688 0.05489919  0.065206114
## [3,] 0.6077575 0.07016951 0.12372173  0.198351208
## [4,] 0.7573788 0.11415508 0.12434936  0.004116792
## [5,] 0.5363323 0.06614164 0.36843126  0.029094802
## [6,] 0.6866064 0.05790994 0.02437947  0.231104142
```

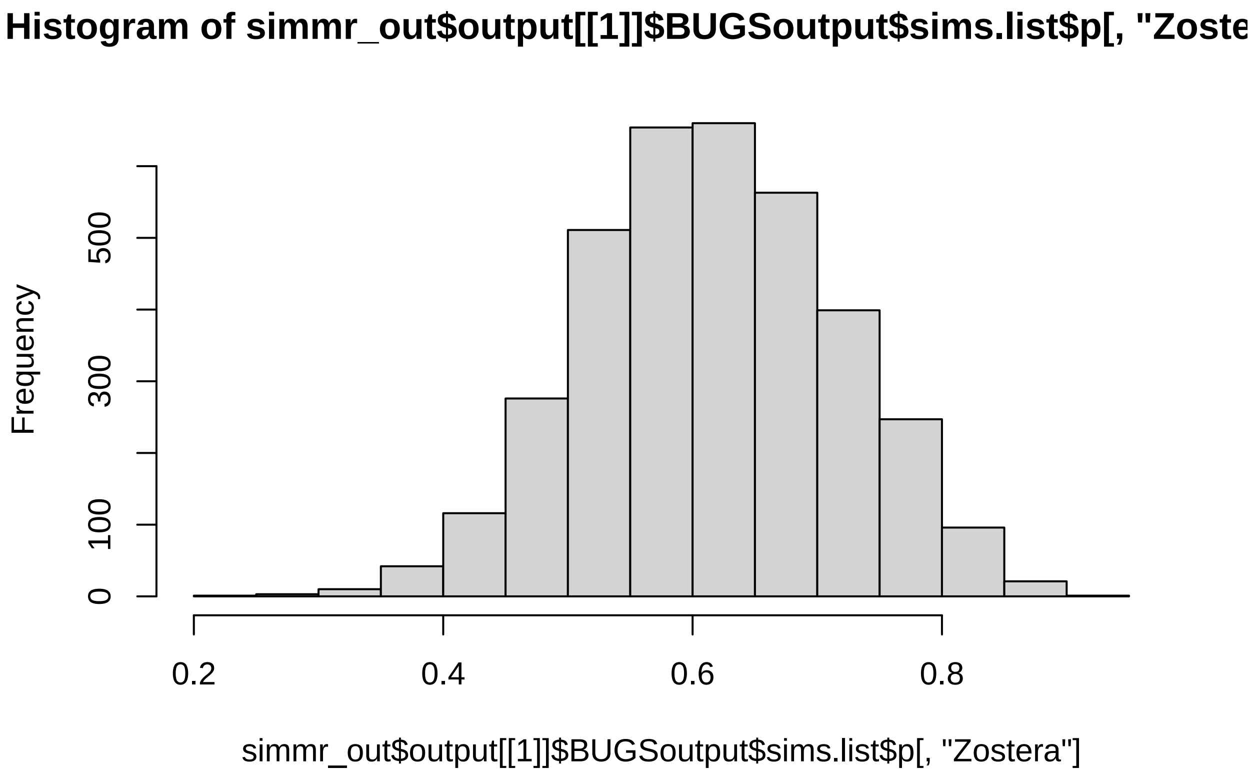
You will see that, for each row, each of the four sources sum to 1:

```
sum(simmr_out$output[[1]]$BUGSoutput$sims.list$p[1,])
```

```
## [1] 1
```

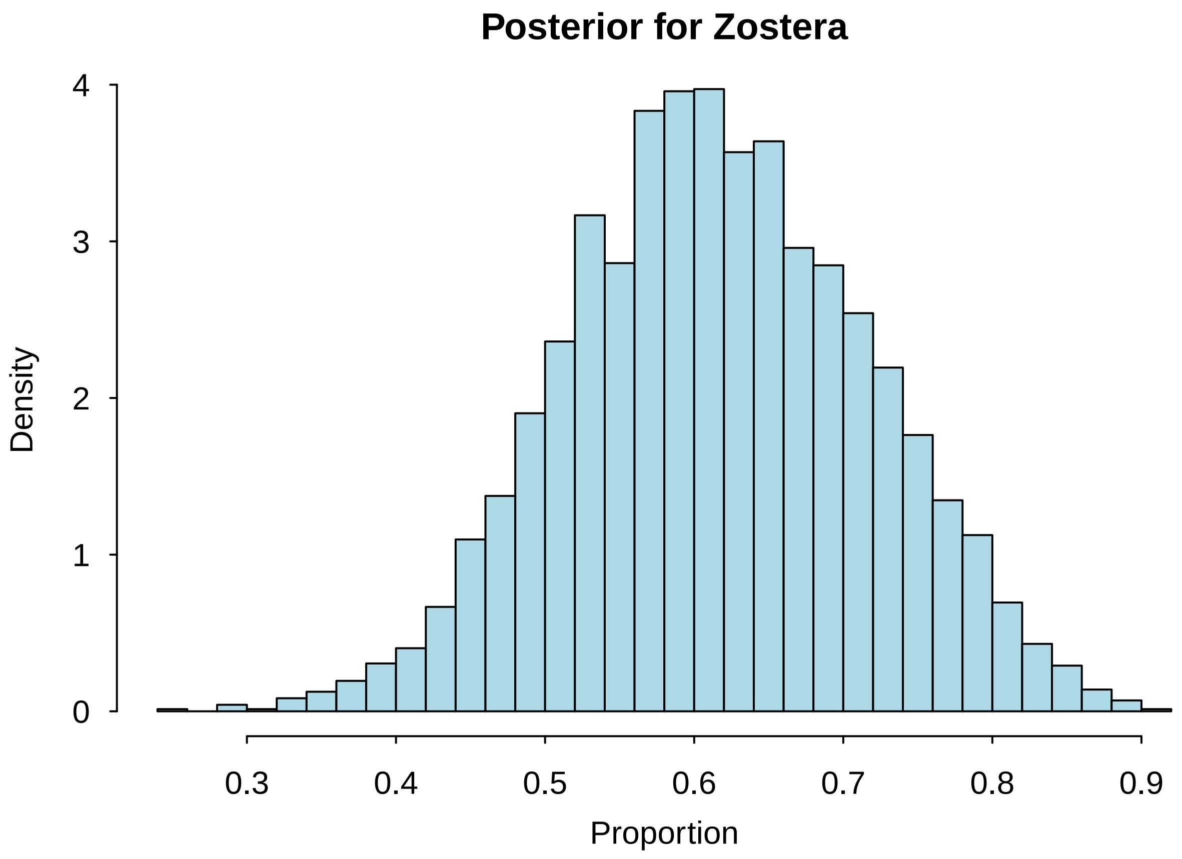
You can now create any further that you wish, for example a simple histogram of the posterior proportion for the first chain of Zostera:

```
hist(simmr_out$output[[1]]$BUGSoutput$sims.list$p[,'Zostera'])
```



This is a bit crude, but with some extra options, you can make this look quite neat:

```
# Set some better options for graphs
par(mar=c(3,3,2,1), mgp=c(2,.7,0), tck=-.01,las=1)
hist(simmr_out$output[[1]]$BUGSoutput$sims.list$p[,'Zostera'],
     freq=FALSE,main='Posterior for Zostera',
     xlab='Proportion',col='lightblue',breaks=30)
```



You can also create your own output analysis. For example, what is the 90% credible interval for Grass?

```
quantile(simmr_out$output[[1]]$BUGSoutput$sims.list$p[,'Grass'],
         probs=c(0.05,0.95))
```

```
##         5%        95%
## 0.03999881 0.11065502
```

What is the probability that the consumers ate more Ulva Lactuca than Enteromorpha?

```
sum(simmr_out$output[[1]]$BUGSoutput$sims.list$p[,'U.lactuca'] >
      simmr_out$output[[1]]$BUGSoutput$sims.list$p[,'Enteromorpha']) /
  nrow(simmr_out$output[[1]]$BUGSoutput$sims.matrix)
```

```
## [1] 0.4352778
```

The above counts the number of rows (i.e. iterations) in the output where Ulva Lactuca is higher than Enteromorpha and divides this by the total number of rows. This is essentially what the `compare_groups` function is doing. There is also a `compare_sources` function which also creates something similar to the above.

Finally, if you want to see what simmr is doing behind the scenes, you can type in the name of the function with the package name separated by three colons, for example

```
simmr:::simmr_mcmc.simmr_input
```

If the simmr plot or table doesn't exactly match what you want you can create your own function based on the original one which includes everything you need.

There is even more detail on the extra functions in simmr in the simmr vignette so make sure to read that before trying to create something complicated yourself.

---

1. Try accessing the output from the second Geese data set (stored above in `simmr_out2`). Try to re-create the above histograms for some of the groups.
2. Continuing the above, try and calculate the probability that one group ate more of a certain source than another.

Some extra tasks

If you finish all the above and want some further tasks to complete try these.

---

1. See if you can re-create the iso-space plot from the raw data from scratch. Refer back to the code in `plot.simmr_input` if you need to.
2. Try and write your own function to process the output from a simmr model run. What would you like to include? Below is a function which just lists the first 15 iterations. You could create something far richer, including means (via `mean`), credible intervals (via `quantile`), correlations (via `cor`) or plots. The `apply` function is often useful here as it will run a function over the rows or columns of a matrix.

```
my_summary = function(x) {
  head(x$output[[1]]$BUGSoutput$sims.matrix,15)
}
my_summary(simmr_out)
```

```
##        deviance   Zostera      Grass  U.lactuca Enteromorpha sd[d13CPl]
##  [1,] 53.53697 0.5872722 0.05757218 0.19916548  0.155990187 0.89530258
##  [2,] 54.93214 0.7991078 0.08078688 0.05489919  0.065206114 0.28209667
##  [3,] 51.54679 0.6077575 0.07016951 0.12372173  0.198351208 0.94063840
##  [4,] 57.88267 0.7573788 0.11415508 0.12434936  0.004116792 1.62488383
##  [5,] 58.23980 0.5363323 0.06614164 0.36843126  0.029094802 1.69514079
##  [6,] 53.46256 0.6866064 0.05790994 0.02437947  0.231104142 1.15908012
##  [7,] 51.51533 0.5787340 0.06369904 0.16877343  0.188793521 0.93774588
##  [8,] 52.31062 0.6986887 0.09060385 0.05764477  0.153062654 0.42968997
##  [9,] 52.65560 0.5846920 0.07157753 0.20777381  0.135956694 0.71577418
## [10,] 53.05297 0.6742526 0.09081898 0.13628034  0.098648039 0.89943344
## [11,] 51.72204 0.5678620 0.08621777 0.19250499  0.153415205 0.86742144
## [12,] 56.22406 0.5823372 0.09131568 0.28404076  0.042306383 0.68843967
## [13,] 51.40634 0.6553609 0.06016441 0.03469594  0.249778761 0.89254363
## [14,] 54.27760 0.4581397 0.05094273 0.08819324  0.402724295 1.08754072
## [15,] 54.57187 0.7958780 0.06599881 0.09376672  0.044356497 0.02341759
##        sd[d15NPl]
##  [1,] 0.581351491
##  [2,] 0.819573019
##  [3,] 0.351675914
##  [4,] 0.745658945
##  [5,] 0.281386513
##  [6,] 0.420286441
##  [7,] 0.150075590
##  [8,] 0.013347580
##  [9,] 0.599871428
## [10,] 0.574419352
## [11,] 0.008292042
## [12,] 1.111461051
## [13,] 0.085674600
## [14,] 0.650509691
## [15,] 0.487712657
```