Practical: Using MixSIAR and JAGS

Andrew Parnell

Introduction

In this practical, we will go through:

- Running some of the examples in MixSIAR
- Creating some simple SIMMs in JAGS
- Creating superior plots of output
- Extending the JAGS SIMMs as discussed in the lectures

This document is an alternative version of the MixSIAR manual (see here) and a much simplified version of our 2013 Environmetrics paper (also available in the Google shared directory).

As in previous labs 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. 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`.

Running MixSIAR

Assuming you have followed the instructions for installing MixSIAR you can run it with:

```
setwd("/path/to/MixSIAR/r")
source("mixsiar_gui.r")
mixsiar_gui()
```

This will bring up the standard MixSIAR gui

At this point you should go back to the module 'Using MixSIAR' and run the Wolves example as detailed in the slides. As part of the model run MixSIAR produces a file `MixSIAR_model.txt` which contains the JAGS script which MixSIAR uses. It's usually a very good idea to have a look at this file to see exactly what model MixSIAR is fitting.

---

1. Go back and run the same examples using the `mixsiar_script.r` file. Check that you understand the output
2. Compare the results with random effects structures. How do the posterior distributions change?
3. The Geese data are also included in MixSIAR, as detailed in the manual in Section 3.4 (P29). run through the Geese example. How does it compare to the SIAR output?

Runnning a basic SIMM in JAGS

Let's start with the simple SIMM defined in the lectures. The model script is as follows:

```
modelstring ='
model {
  for(i in 1:N) {
    for(j in 1:J) {
      y[i,j] ~ dnorm(inprod(p*q[,j],s[,j]+c[,j])/inprod(p,q[,j]),1/pow(sigma[j],2))
    }
  }
  for(k in 1:K) {
    for(j in 1:J) {
      s[k,j] ~ dnorm(s_mean[k,j],s_sd[k,j]^-2)
      c[k,j] ~ dnorm(c_mean[k,j],c_sd[k,j]^-2)
    }
  }
  p ~ ddirch(alpha)
  for(j in 1:J) { sigma[j] ~ dunif(0,10) }
}
'
```

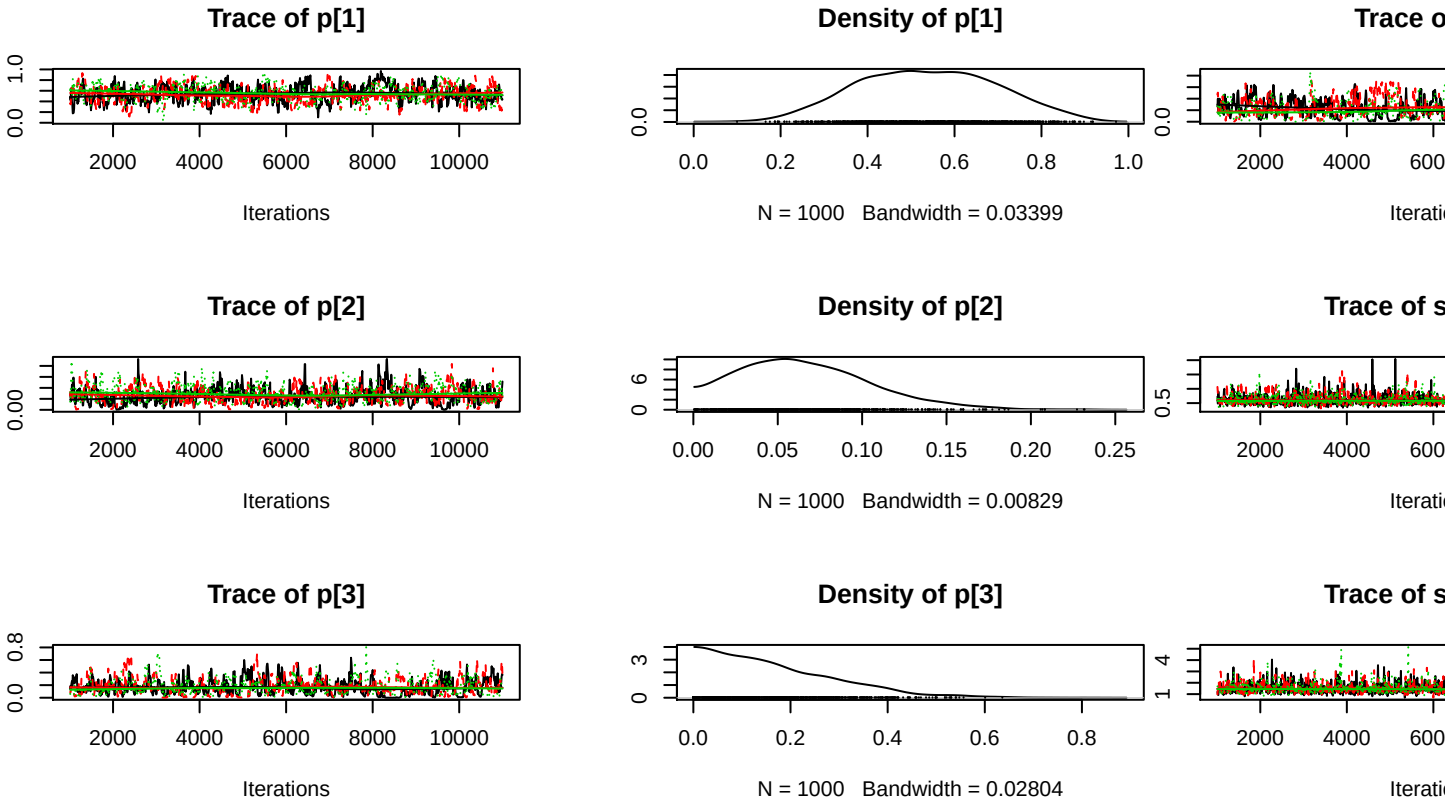
Make sure you understand what each of the lines of this code is doing. You can run it on the basic Geese data with:

```
library(siar)
library(rjags)
```

```
data(geese1demo,sourcesdemo,correctionsdemo,concdepdemo)
con = geese1demo
sources = as.matrix(sourcesdemo[,2:5])
tefs = as.matrix(correctionsdemo[,2:5])
cd = as.matrix(concdepdemo[,c(2,4)])
data=list(y=con,s_mean=sources[,c(1,3)],s_sd=sources[,c(2,4)],
          c_mean=tefs[,c(1,3)],c_sd=tefs[,c(2,4)],
          q=cd,N=nrow(con),K=nrow(sources),
          J=ncol(con),alpha=rep(1,nrow(sources)))
model_1=jags.model(textConnection(modelstring), data=data,n.chain=3)
output=coda.samples(model=model_1,variable.names=c('p','sigma'),n.iter=10000,thin=10)
```

We've already plotted the isospace plot before with SIAR, so let's check convergence:

```
plot(output)
```




```
gelman.diag(output,multivariate=FALSE)
```

```
## Potential scale reduction factors:
##
##          Point est. Upper C.I.
## p[1]           1.00       1.02
## p[2]           1.01       1.01
## p[3]           1.01       1.03
## p[4]           1.00       1.00
## sigma[1]       1.00       1.00
## sigma[2]       1.00       1.00
```

This looks reasonable to me. Most of the trace plots are stable and the density plots look sensible. The Gelman diagnostics are pretty close to 1 (maybe it needs just a slightly longer run to get them right down to 1). Note that when dealing with a SIMM you need to add the extra argument `multivariate=FALSE` to the Gelman diagnostics because the parameters are highly dependent (because the proportions sum to 1).

Let's now get some results from the output. Start with a simple table of results:

```
round(summary(output)$quantiles,3)
```

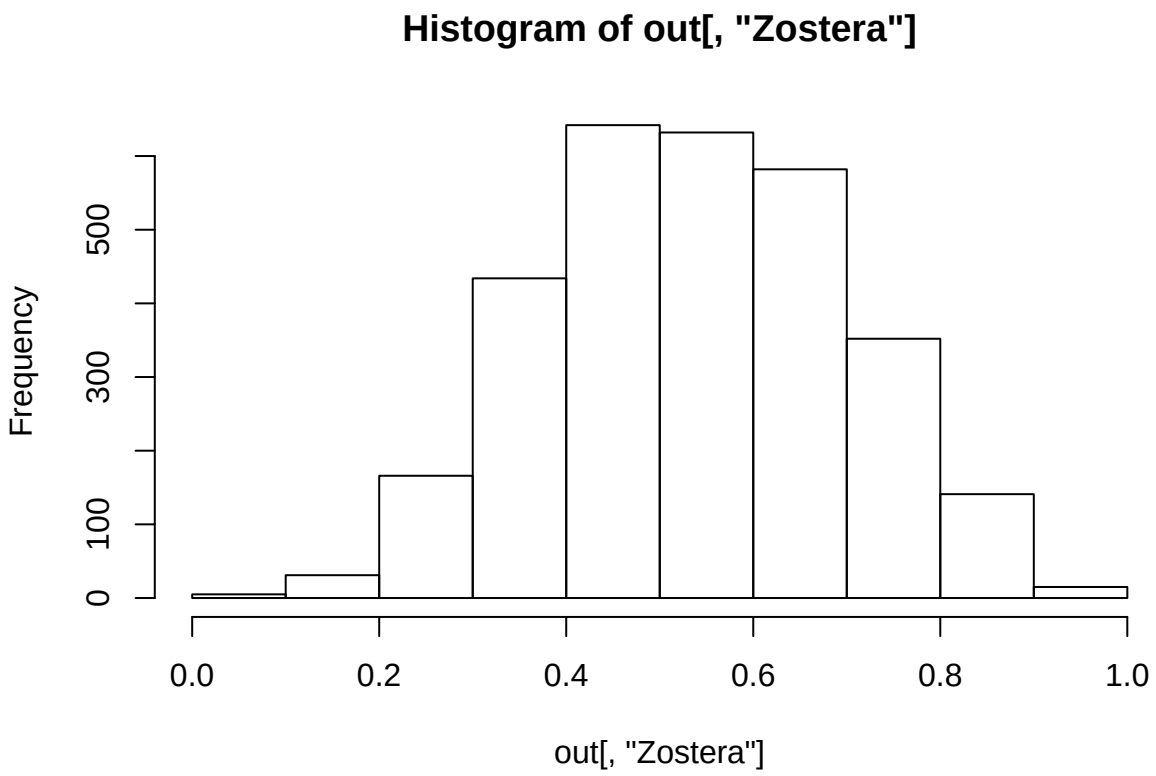
```
##           2.5%   25%   50%   75% 97.5%
## p[1]     0.243 0.418 0.537 0.654 0.839
## p[2]     0.006 0.038 0.062 0.091 0.153
## p[3]     0.006 0.064 0.143 0.250 0.488
## p[4]     0.006 0.089 0.192 0.332 0.615
## sigma[1] 0.365 0.482 0.576 0.702 1.098
## sigma[2] 0.912 1.194 1.436 1.733 2.739
```

These should be similar, but not identical to, the SIAR results from practical 2. (Remember the SIAR model is slightly inferior to this JAGS model). You can get a fuller description of the output with `summary(output)`

and also store the results from this in an object to access different parts (see `str(summary(output))`).

We can create histograms of the different proportions via

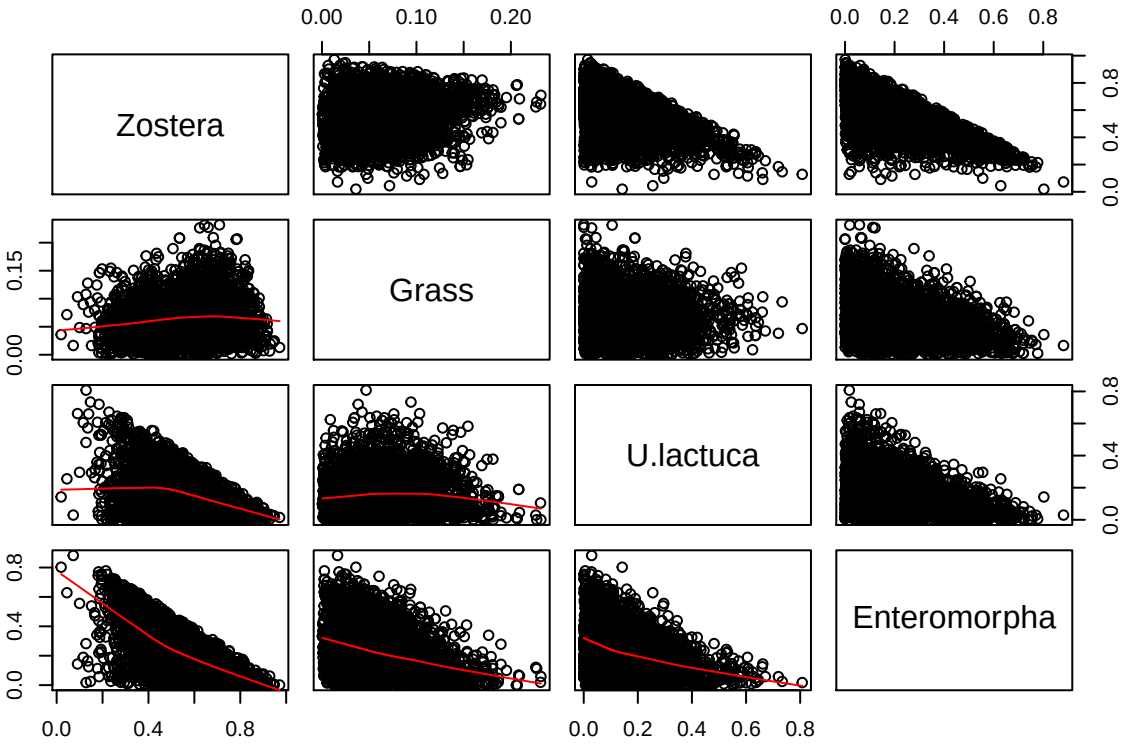
```
out = do.call(rbind,output)
colnames(out) = c(as.character(sourcesdemo[,1]),'sd1','sd2')
hist(out[,'Zostera'])
```



The `do.call` command above takes the list of different chains and combines them into a single big matrix for ease of analysis. Compare the structure (via `str`) of `output` and `out` if you want to learn more.

Finally, we can get a matrix plot with

```
pairs(out[,1:4],lower.panel=panel.smooth)
```



---

1. Try playing with the JAGS script. Change the priors, change the data (e.g. remove a source), increase the number of iterations, etc
2. Can you make the histograms of the different proportions look nicer? Can you plot multiple histograms (i.e. for different sources) on the same plot?
3. Have a look at the help file for the `pairs` function. See if you can change the code above to create a better SIAR-type matrix plot with histograms on the diagonal and correlations in the upper diagonal.

More advanced SIMMs in JAGS

Let's now take the second more advanced SIMM from the module on complex SIMMs and run this on the Geese data. The new JAGS code is:

```
modelstring ='
model {
  for (i in 1:N) {
    for (j in 1:J) {
      y[i,j] ~ dnorm(inprod(p[i,]*q[,j], s_mean[,j]+c_mean[,j]) / inprod(p[i,],q[,j]), 1/var_y[i,j])
      var_y[i,j] <- inprod(pow(p[i,]*q[,j],2),s_sd[,j]^2+c_sd[,j]^2)/pow(inprod(p[i,],q[,j]),2)
        + pow(sigma[j],2)
    }
  }
  for(i in 1:N) {
    p[i,1:K] <- expf[i,]/sum(expf[i,])
    for(k in 1:K) {
      expf[i,k] <- exp(f[i,k])
      f[i,k] ~ dnorm(mu_f[k],sigma_f[k]^-2)
    }
```

```
  }
  for(k in 1:K) {
    mu_f[k] ~ dnorm(0,1)
    sigma_f[k] ~ dgamma(2,1)
  }
  for(j in 1:J) { sigma[j] ~ dunif(0,10) }
}
'
```

See if you can follow all of the different parts. If not, ask me to help explain this in more detail. We can run this model on the Geese data with:

```
data=list(y=con,s_mean=sources[,c(1,3)],s_sd=sources[,c(2,4)],
          c_mean=tefs[,c(1,3)],c_sd=tefs[,c(2,4)],
          q=cd,N=nrow(con),K=nrow(sources),
          J=ncol(con))
model_2=jags.model(textConnection(modelstring), data=data,n.chain=3)
output=coda.samples(model=model_2,variable.names=c('mu_f','sigma_f','sigma'),n.iter=10000,thin=10)
```

The convergence diagnostics are

```
gelman.diag(output,multivariate=FALSE)
```

```
## Potential scale reduction factors:
##
##            Point est. Upper C.I.
## mu_f[1]          1.01       1.05
## mu_f[2]          1.02       1.06
## mu_f[3]          1.00       1.01
## mu_f[4]          1.01       1.02
## sigma[1]         1.00       1.00
## sigma[2]         1.00       1.00
## sigma_f[1]       1.01       1.02
## sigma_f[2]       1.01       1.02
## sigma_f[3]       1.06       1.11
## sigma_f[4]       1.01       1.03
```

I haven't included `plot(output)` here as it gets a bit long, however you might want to run it to confirm the model has reasonably well converged. As you can in the `output=` statement above I haven't saved the proportions `p` though you can change this if you wish. Instead we'll just look at the overall proportions defined through `mu_f`. We can get at these via inverse CLR transform of each iteration.

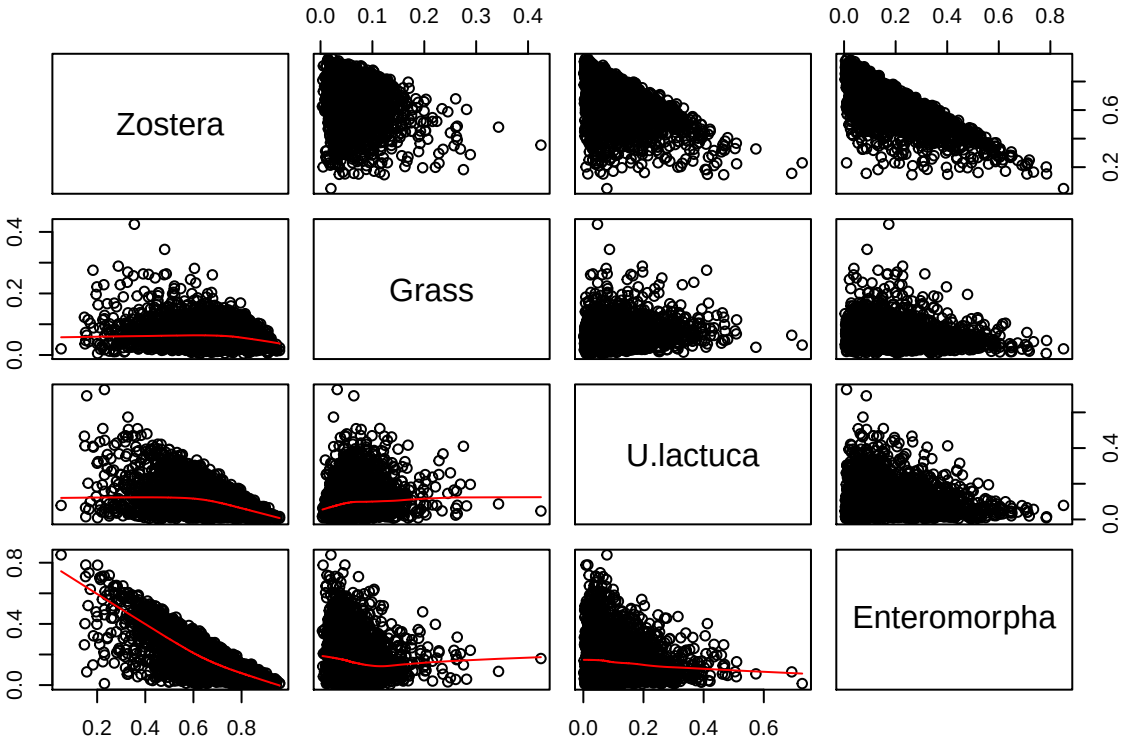
```
library(compositions)
```

```
# Create matrix of all chains
out = do.call(rbind,output)
# Apply the CLR inverse transform to make them into proportions
p_mean = t(apply(out[,1:4],1,clrInv))
colnames(p_mean) = sourcesdemo[,1]
# Now can perform same analysis as before
t(round(apply(p_mean,2,'quantile',probs=c(0.025,0.5,0.975)),3))
```

```
##               2.5%   50% 97.5%
## Zostera      0.318 0.670 0.881
## Grass        0.017 0.060 0.153
## U.lactuca    0.012 0.085 0.327
## Enteromorpha 0.017 0.138 0.496
```

and a matrix plot

```
pairs(p_mean,lower.panel=panel.smooth)
```



---

1. Try changing the priors on `sigma_f` and `mu_f`. Do they have much of an effect on the posterior?
2. Run the same model on some of the `geese2demo` groups. You can create a new consumers file by first running `data(geese2demo)` and then e.g. `new_con = subset(geese2demo[,2:3],geese2demo[,1]==2)`. Check that you can understand the output produced by the JAGS model

---

Comparing models

To compare whether the simpler or more complicated SIMM works best, let's compute the DIC (see the DIC lecture notes for details). We can do this with two simple commands. For model 1:

```
dic.samples(model=model_1,n.iter=10000,thin=10,type='popt')
```

```
## Mean deviance:  46.56
## penalty 14.05
## Penalized deviance: 60.61
```

and for model 2:

```
dic.samples(model=model_2,n.iter=10000,thin=10,type='popt')
```

```
## Mean deviance:  53.81
## penalty 40.48
## Penalized deviance: 94.29
```

The DIC values suggest that the simpler model is better for the Geese data as it has a smaller DIC. Note however that the deviance, a raw measure of model fit irrespective of complexity, suggests that the second

model fits the data better. The smaller DIC for the simpler model is perhaps not surprising given the size and complexity of the data set. Note that I've used the 'popt' version of DIC here as this tends to work better for complex models.

---

1. Try changing the values of `alpha` in the simpler model and compare the DIC values. Do they change much? What happens if you put use alpha values that disagree with the data? (e.g. a relatively small alpha value for Zostera)
2. Try changing the `type` in the `dic.samples` command above for the simpler model. Does the penalty roughly match the true number of parameters?

Further challenges

If you have completed all of the above try the below harder challenges

---

1. Associated with the modules on complex SIMMs are some R scripts which run the harmonic and spline models on the larger Geese data set. Run the fitting and plotting scripts and see if you can understand what is being run. If the JAGS runs are too slow on your computer use my saved version of the output (this has an `.rda` file extension)
2. We previously used the multivariate normal distribution on the isotope data where, at least for the simple geese data, it didn't seem to have much effect. Try and adapt the more complex SIMM used in this practical to have a multivariate normal prior on the CLR transformed proportions. Use the Wishart distribution as a prior on the covariance matrix
3. MixSIAR contains some further data sets (e.g. the Lake and Wolves data). See if you can fit these models using the JAGS scripts outlined in this course and get similar answers to the MixSIAR model. You might like to refer back to the MixSIAR JAGS code which is created as part of the MixSIAR model run

---