Module 7: An introduction to MixSIAR

Andrew Parnell, School of Mathematics and Statistics,
University College Dublin

Learning outcomes

- Random effects models
- Understand how MixSIAR extends MixSIR and SIAR
- Understand the differences in likelihoods and priors
- Be able to understand output from MixSIAR

Revision: SIAR model in JAGS

- Let's move back to the proper SIAR model defined yesterday:

$$y_{ij} \sim N\left(\frac{\sum_{k=1}^{K} p_k q_{jk}(\mu_{s,jk} + \mu_{c,jk})}{\sum_{k=1}^{K} p_k q_{jk}}, \frac{\sum_{k=1}^{K} p_k^2 q_{jk}^2(\sigma_{s,jk}^2 + \sigma_{c,jk}^2)}{(\sum_{k=1}^{K} p_k q_{jk})^2} + \sigma_j^2\right)$$

- We also have prior distributions (usually uniform) on σ^2 and a Dirichlet prior on p
- How is this model still a simplification of reality?

Expanding the SIAR model further

Some (of the many) possible extensions:

1. We are assuming that all consumers have identical dietary proportions
2. We are assuming that residuals, sources and TEFs are uncorrelated across isotopes
3. We are assuming that concentration dependence is known
4. We cannot add in any extra covariates (height, weight, etc, etc)

Mixed effects models in linear regression

- Often data are available in *groups*, for example wolves might belong to different packs
- We want to capture the different levels of variation, both *within* groups, and *between* groups
- Example: suppose y_{ij} is a measurement for individual i in group j, $i = 1, \ldots, N_j$, $j = 1, \ldots, M$
- We might use a model such as:

$$y_{ij} \sim N(\mu + b_j, \sigma^2), \; b_j \sim N(0, \sigma_b^2)$$

- Now b_j is called a *random effect* and measures the change in the mean for each group
- σ measures the standard deviation *within* a group whilst σ_b measures the standard deviation *between* groups

Fitting a random effects model in JAGS

```
modelstring ='
model {
  for(i in 1:N) { y[i] ~ dnorm(mu+b[group[i]],1/pow(sigma,2
  for(j in 1:M) { b[j] ~ dnorm(0,1/pow(sigma_b,2)) }
  mu ~ dnorm(0,0.001)
  sigma ~ dunif(0,100)
  sigma_b ~ dunif(0,10)
}'
data=list(y=c(3.03, 2.68, 2.04, 3.23, 2.82, 2.46, 3.06, 3.0
              3.02, 2.95, 3.12, 2.81, 2.69, 2.65, 2.49, 2.7
              2.21, 3.92, 4.7, 3.53, 3.89, 3.86, 4.48),
          group=c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2,
                  2, 2, 2, 3, 3, 3, 3, 3),
          N=23,M=3)
model=jags.model(textConnection(modelstring), data=data)
output=coda.samples(model=model,
                    variable.names=c("mu","sigma","sigma_b"
                    n.iter=1000)
```

Output from the random effects model

```
t(round(apply(output[[1]],2,quantile,probs=c(0.025,0.5,0.97
```

```
##             2.5%  50% 97.5%
## mu          0.23 3.22  5.96
## sigma       0.35 0.46  0.67
## sigma_b     0.36 1.22  6.12
```

- You need a reasonable number of groups to estimate σ_b well. When the number of groups is very small you might run into problems
- You could also include `b` in the `variable.names` if you want the deviations from the overall mean
- Convergence is often better if you *hierarchically centre* the model, which means set $b_j \sim N(\mu, \sigma_b^2)$ in the previous slides' JAGS code (if you look in the code for these slides that is what I ran)

Defining mixed effects models in the SIMM case

- In the SIMM case we might have information that some consumers might share the same dietary proportions
- We might be interested in how the dietary proportions vary between groups and within groups, just as in the simple example
- An issue is: how do we achieve this in the SIMM case?
- One solution was elegantly proposed by Semmens *et al* in their PLoS ONE 2009 paper

A hierchical SIMM

- Semmens *et al* differentiate between the *overall* or *mean* dietary proportions p_{mean}, and the *deviations* $p_{dev,j}$ from this mean for each *group* p_j
- They use the relationship:

$$p_j = wp_{mean} + (1 - w)p_{dev,j}$$

  where w is a mixing constant which determines how much the group behaves like the mean (high w), or not (low w)
- They give the mean and the deviation dietary proportions Dirichlet$(\alpha_1, \ldots, \alpha_K)$ prior distributions
- They give w a standard $U(0, 1)$ prior

More on random effects in proportions

- A slightly more flexible prior distribution for proportions is obtained by transforming the proportions instead
- We already met this with logistic regression where we can use:

$$f = \mathit{logit}(p) = \log\left(\frac{p}{1-p}\right) \text{ or equivalently } p = \frac{\exp(f)}{\exp(f)+1}$$

- When we have multiple proportions a generalisation of this is the *centralised log ratio* (CLR) or *softmax* transformation:

$$[p_1, \ldots, p_K] = \left[\frac{\exp(f_1)}{\sum_j \exp(f_j)}, \ldots, \frac{\exp(f_K)}{\sum_k \exp(f_k)}\right]$$

The CLR transformation

- In logistic regression we can put a prior distribution on f (i.e. $logit(p)$) e.g. $f \sim N(\alpha + \beta x, \sigma^2)$ which allows us to relate the probability p to a covariate x. We can use the normal distribution because f is unrestricted
- In CLR regression, we put a prior on the f_k so that each one relates to the covariate with different coefficient values
- The CLR transformation guarantees that all the dietary proportions will sum to 1
- More on this tomorrow

Random effects for individuals

- We don't necessarily need a grouping structure (e.g. pack, sex, etc) to be able to include random effects in a SIMM
- In a SIMM we might reasonably assume that every consumer is eating something slightly different and want to quantify the overall mean diet as well as the variability between consumers
- We can do this by modelling each consumer's dietary proportion p_{ik} with a normally distributed prior on the CLR transform of p

A 'simple' CLR example

```
modelstring ='
model {
  for (j in 1:J) {
    for (i in 1:N) {
      y[i,j] ~ dnorm(inprod(p[i,]*q[,j], s_mean[,j]+c_mean
      var_y[j] <- inprod(pow(p[i,]*q[,j],2),1/s_prec[,j]+1/
    }
  }
  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],1/pow(sigma_f[k],2))
    }
  }
  for(k in 1:K) {
    mu_f[k] ~ dnorm(0,1)
    sigma_f[k] ~ dgamma(2,1)
```

CLR model: R code

```
data=list(y=con,s_mean=sources[,c(1,3)],s_prec=1/sources[,c
          c_mean=tefs[,c(1,3)],c_prec=1/tefs[,c(2,4)]^2,
          q=cd,N=nrow(con),K=nrow(sources),
          J=ncol(con))
model=jags.model(textConnection(modelstring), data=data)
output=coda.samples(model=model,variable.names=c('p','sigma
                    n.iter=10000)
out_summ = summary(output)
```

Output

```
head(out_summ$statistics,12)
```

```
##                  Mean        SD    Naive SE Time-series SE
## mu_f[1]   1.5725428 0.6805714 0.006805714    0.047989801
## mu_f[2] -0.8027497 0.6839776 0.006839776    0.049487668
## mu_f[3] -0.4957031 0.8491425 0.008491425    0.047860117
## mu_f[4] -0.1392164 0.9283296 0.009283296    0.045838615
## p[1,1]   0.6660175 0.1923927 0.001923927    0.005102431
## p[2,1]   0.6285213 0.2032308 0.002032308    0.005140367
## p[3,1]   0.6804065 0.1937963 0.001937963    0.005357839
## p[4,1]   0.6476960 0.2007856 0.002007856    0.005324347
## p[5,1]   0.6523277 0.1950306 0.001950306    0.005310548
## p[6,1]   0.6815461 0.1924507 0.001924507    0.004930912
## p[7,1]   0.6973124 0.1830500 0.001830500    0.005092312
## p[8,1]   0.4558782 0.2190748 0.002190748    0.007828994
```

Notes about the CLR model

- This is a great starter script for your own work. If you can understand this code and adapt it to your data you can get some really powerful results
- We can now put covariates in the model: we just have to expand μ_f in the previous JAGS code
- We now have individual dietary proportion estimates (p_{ik}) and overall dietary proportion estimates (via CLR transform of μ_k), and also estimates of the variability (from σ_f)
- Things start to get complicated with prior distributions at this stage. Be very careful and always examine prior sensitivity by re-running the model with slightly different prior distributions. Look at the effect on p (the effect on μ_f and σ_f is less important)

Back to MixSIAR

- MixSIAR implements a version of the CLR jags code already shown, and through this allows the introduction of random effects and covariates
- It's a bit limited in that:
  1. It's recommended to work with a GUI
  2. You can only put a certain number of random effects/covariates into the model
  3. It doesn't currently take account of uncertainty in TEFs (it calls this *discrimination*)
  4. It doesn't allow for much choice in prior distributions
- However, it's a great start at finding simple ways to run more complicated models

The main MixSIAR input screen

Running the Wolves example

- In the MixSIAR folder there are three files: `wolves_consumer.csv`, `wolves_dicrimination.csv`, and `wolves_sources.csv`
- These contain the consumer isotope data with two covariates, the discrimination data (no standard deviations), and the source data (for each different region)
- Note that MixSIAR allows you to input either the sources means/sds (as we have been using), or the raw source data, to which it fits a model to estimate the source parameters
- These data are similar to those used in the Semmens *et al* PLoS ONE paper already mentioned. The methods are more fully described in that paper
- The variables pack and region here are included as *nested* random effects, meaning that there are two layers of random effects which measure variability between regions and variability between packs. The overall residual measures variability within pack

Wolves example - consumers

Wolves example - sources

Wolves example - isospace plot

Wolves example - running

- First run the model as a test go, then choose a normal run
- These models are starting to get complicated, they take a while to run
- MixSIAR write JAGS code on the fly. Check out the JAGS model structure in `MixSIAR_model.txt`
- When it's finished click on `process output` to get at the (many) results
- You can also access an R script version of this (much better) in `mixsiar_script.r`

Analysing output

- First, check convergence using Brooks-Gelman-Rubin or Geweke (both previously covered)
- You then have access to histograms of the posterior distributions of the overall means, and also for each level of the hierarchy; here pack and region
- They also produce a pairs plot (like the matrix plot of yesterday) which can tell you about model inadequacy
- Use the `mixsiar_script.r` if you want full access to the JAGS output for any further analysis

MixSIAR output - trace plot

MixSIAR output - random effect densities

MixSIAR output - overall proportion densities

MixSIAR output - matrix plot

Summary

- We have looked at the differences between SIAR and MixSIAR
- This has included: random effects, the centralised log ratio
- We have run one of the more complicated MixSIAR examples