Differences between regression models and SIMMs

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

Learning outcomes

- Be able to describe the differences and similarities between a regression model and a SIMM
- Understand the likelihood and prior distribution in a basic SIMM
- Know how to check convergence and model performance in a Bayesian model

Revision: linear regression

- In many statistical problems we have a *response variable* y_i observed on individuals $i = 1, \ldots, N$
- We also have an *explanatory variable* x_i from which we want to predict y_i
- For example, y_i could be the weight of an animal, and x_i could be the proportion of a certain food source in its diet

The usual linear regression model is written as:

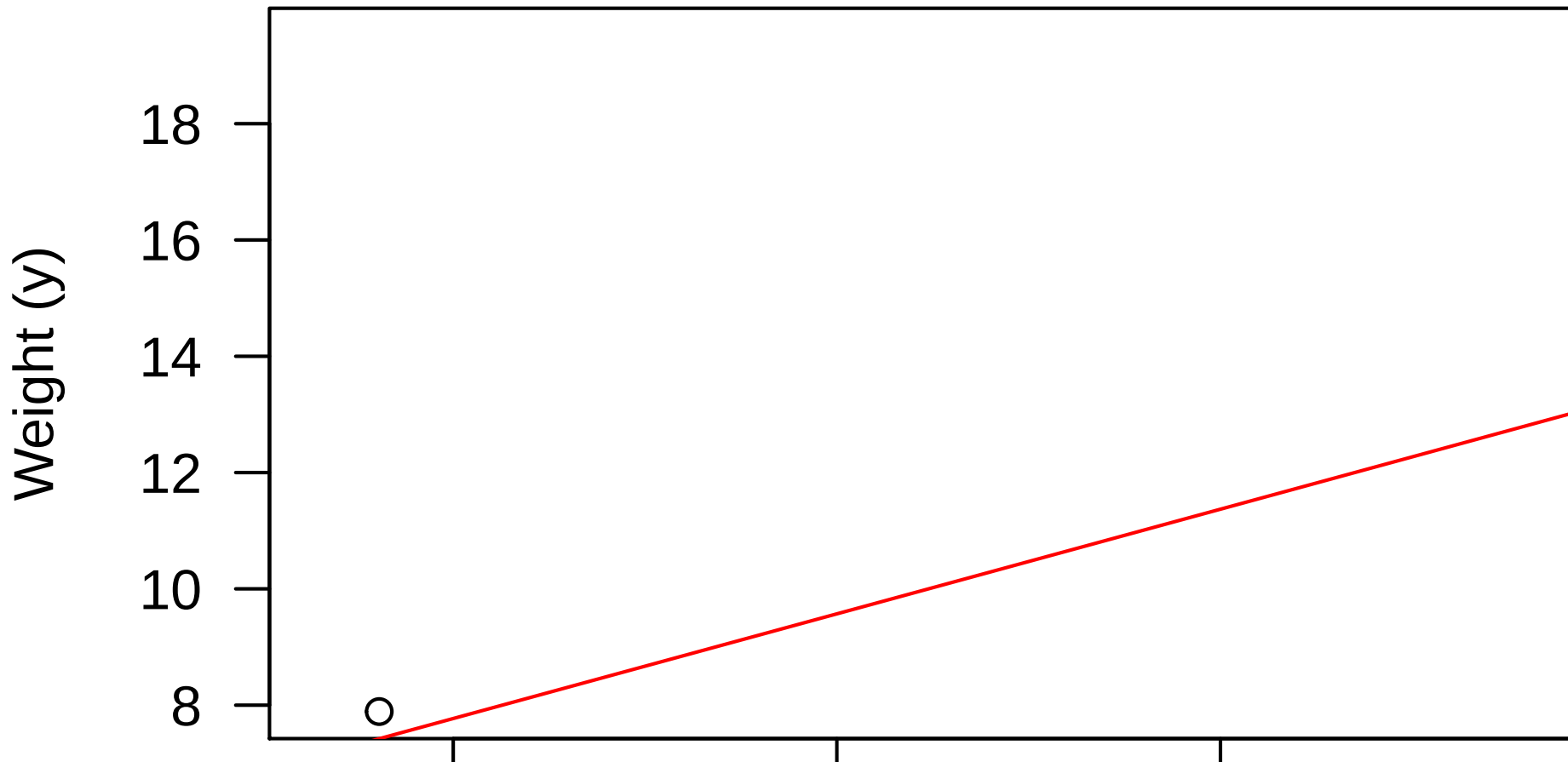
$$y_i = \alpha + \beta x_i + \epsilon_i$$

where $\epsilon_i \sim N(0, \sigma^2)$. Another way of writing this model is:

$$y_i \sim N(\alpha + \beta x_i, \sigma^2)$$

Example: simple data

```
x=c(18.07, 52.59, 54.93, 79.31, 89.58)
y=c(7.89, 12.41, 13.34, 19.3, 19.52)
plot(x,y,xlab='Percentage of food source in diet (x)',ylab=
abline(a=4.17,b=0.18,col='red')
```

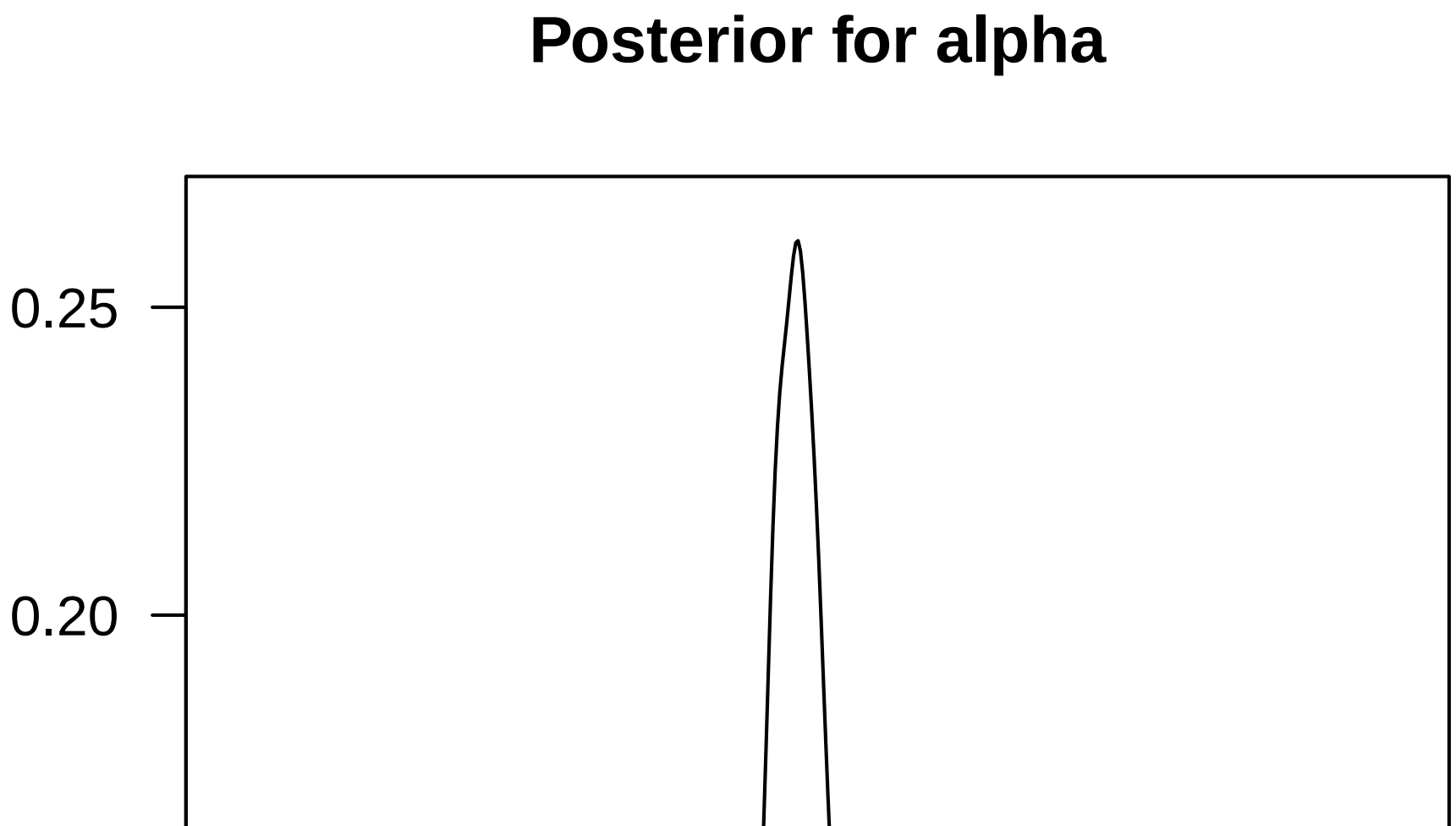


Running a linear regression in JAGS

```
modelstring ='
model {
  for(i in 1:N) {
    y[i] ~ dnorm(alpha + beta*x[i],sigma^-2)
  }
  alpha ~ dnorm(0,100^-2) # Note: these are all vague prior
  beta ~ dnorm(0,100^-2)
  sigma ~ dunif(0,100)
}
'
data=list(x=c(18.07, 52.59, 54.93, 79.31, 89.58),
          y=c(7.89, 12.41, 13.34, 19.3, 19.52),
          N=5)
model=jags.model(textConnection(modelstring), data=data)
output=coda.samples(model=model,variable.names=c("alpha","b
                    n.iter=10000)
```

Output from linear regression

```
par(mfrow=c(1,2),las=1)
plot(density(output[[1]][,'alpha']),main='Posterior for alp
plot(density(output[[1]][,'beta']),main='Posterior for beta
```



More output from linear regression

```
summary(output)$statistics
```

```
##               Mean          SD     Naive SE Time-series SE
## alpha 4.2380578 2.29579268 0.0229579268    0.072311840
## beta  0.1739183 0.03626873 0.0003626873    0.001143307
```

```
summary(output)$quantiles
```

```
##                2.5%       25%       50%       75%     97.5%
## alpha -0.17863572 3.1365954 4.1943201 5.2683452 9.031500
## beta   0.09743827 0.1579924 0.1751919 0.1916921 0.242404
```

How do we choose a likelihood and a prior for this situation?

- When we're in a standard linear regression situation the likelihood is always a normal distribution. When we use other likelihoods we're running a *generalised linear model*
- The prior for the intercept (α) and slope (β) might come from previous experiments, or in this case are set as vague. Similarly for the residual standard deviation
- Sometimes *re-parameterising* the model will help with setting the priors. For example, it might be easier to re-write the model as $y_i = \alpha + \beta(x_i - \bar{x}) + \epsilon_i$. Now the parameter α represents the mean value of y at the mean value of x (denoted $\bar{x}$). This might be easier to put a prior distribution on

Example 2: a generalised linear model situation, e.g. Logistic regression

- Suppose now that rather than observing y as the weight of the animal, we have observed y as whether or not the animal was male ($y_i = 1$) or female ($y_i = 0$)
- The goal of the model is now to estimate the relationship between dietary proportion and the probability of being male.
- When the response variable is binary we use a GLM called *logistic regression* We can write this new model as:

$$y_i \sim Bin(1, p_i),\ logit(p_i) = \alpha + \beta x_i$$

where $logit(p) = \log\left(\frac{p}{1-p}\right)$. Note that p_i directly measures the probability of each individual being male and has to lie between 0 and 1

Example 2 in JAGS

Note: this isn't a great model as the data set is very small

```
modelstring ='
model {
  for(i in 1:N) {
    y[i] ~ dbin(p[i],1)
    logit(p[i]) <- alpha + beta*x[i]
  }
  alpha ~ dnorm(0,100^-2)
  beta ~ dnorm(0,100^-2)
}
'
data=list(x=c(18.07, 52.59, 54.93, 79.31, 89.58),
          y=c(0,1,0,1,1),
          N=5)
model=jags.model(textConnection(modelstring), data=data)
output=coda.samples(model=model,variable.names=c("alpha","b
                    n.iter=10000)
```

Moving on to SIMMs - what do the data look like?

- Let's start with a very simple version:
  - 1 isotope
  - 2 food sources
  - 9 consumers
  - No other complications
- We'll use some of the Geese data from the SIAR package
- Reminder: you can install the up-to-date version of SIAR with:

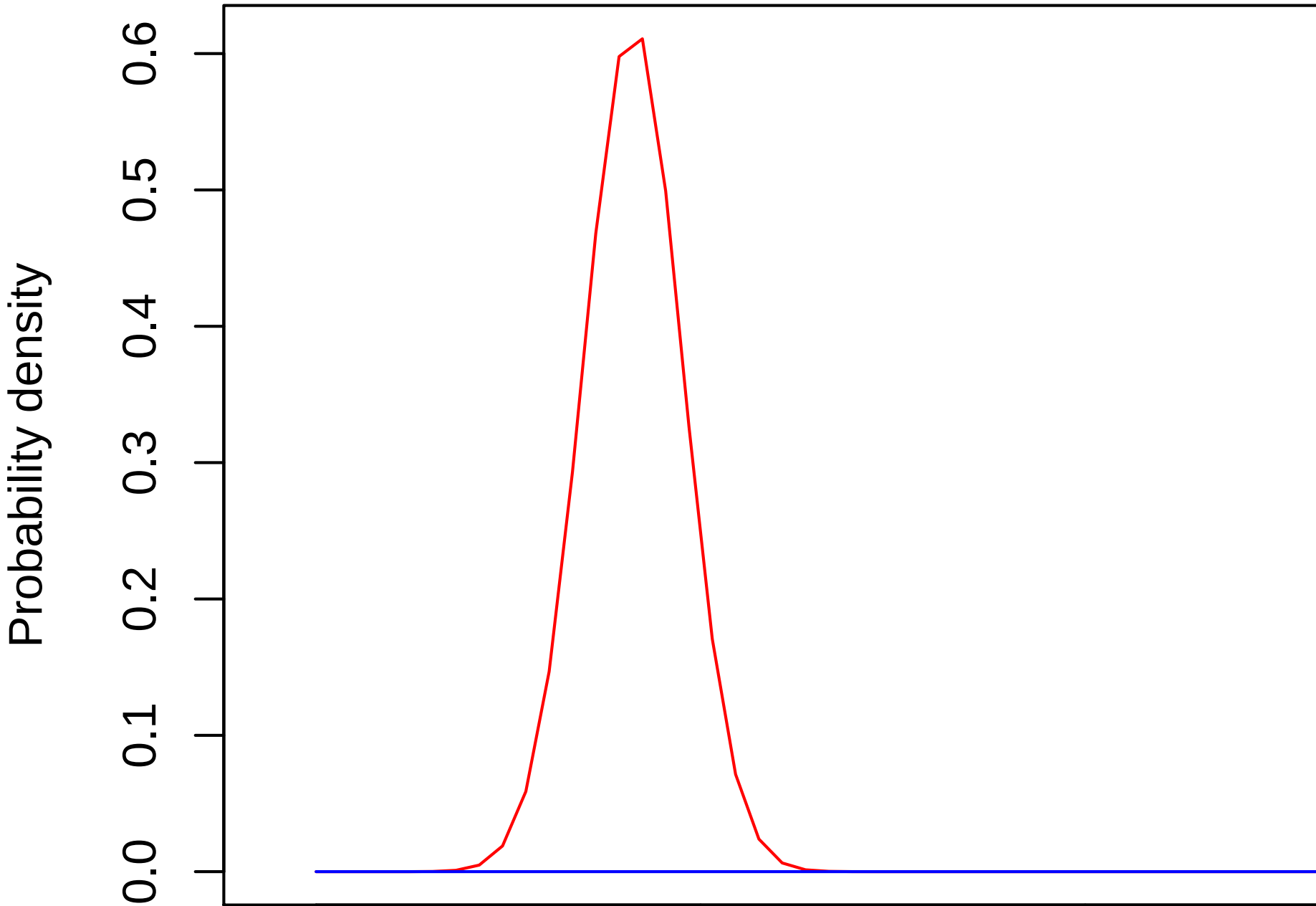
```
library(devtools)
install_github("andrewljackson/siar")
library(siar)
```

Plotting at the data

- Use the second isotope (δ^{13}C) and the first two food sources (Zostera and Grass)
- Create a plot:

```
# Load in the data
data(geese1demo); data(sourcesdemo)
consumers = geese1demo[,2]
sources = sourcesdemo[1:2,4:5]
con_grid = seq(-35,-5,length=100)
plot(con_grid,dnorm(con_grid,mean=sources[2,1],sd=sources[2
     type='l',col='red',xlab='d13C',ylab='Probability densi
lines(con_grid,dnorm(con_grid,mean=sources[1,1],sd=sources
      col='blue')
points(consumers,rep(0,9))
legend('topright',legend=c('Grass','Zostera','Consumers'),
       lty=c(1,1,-1),pch=c(-1,-1,1),col=c('red','blue','bla
```

A simple isospace plot



A first model for this simple SIMM

- Let y_i be the δ^{13}C value for individual i, $i = 1, \ldots, 9$
- Let s_k be the source value for source k, $k = 1, 2$
- Let p_k be the dietary proportion for source k

The likelihood can now be written as:

$$y_i = p_1 \times s_1 + p_2 \times s_2 + \epsilon_i \text{ or } y_i \sim N\left(\sum_{k=1}^{2} p_k s_k, \sigma^2\right)$$

so just like a regression model with a slightly different mean!

$\epsilon_i \sim N(0, \sigma^2)$ as usual, though including this term is (strangely) controversial

Prior distributions for the SIMM

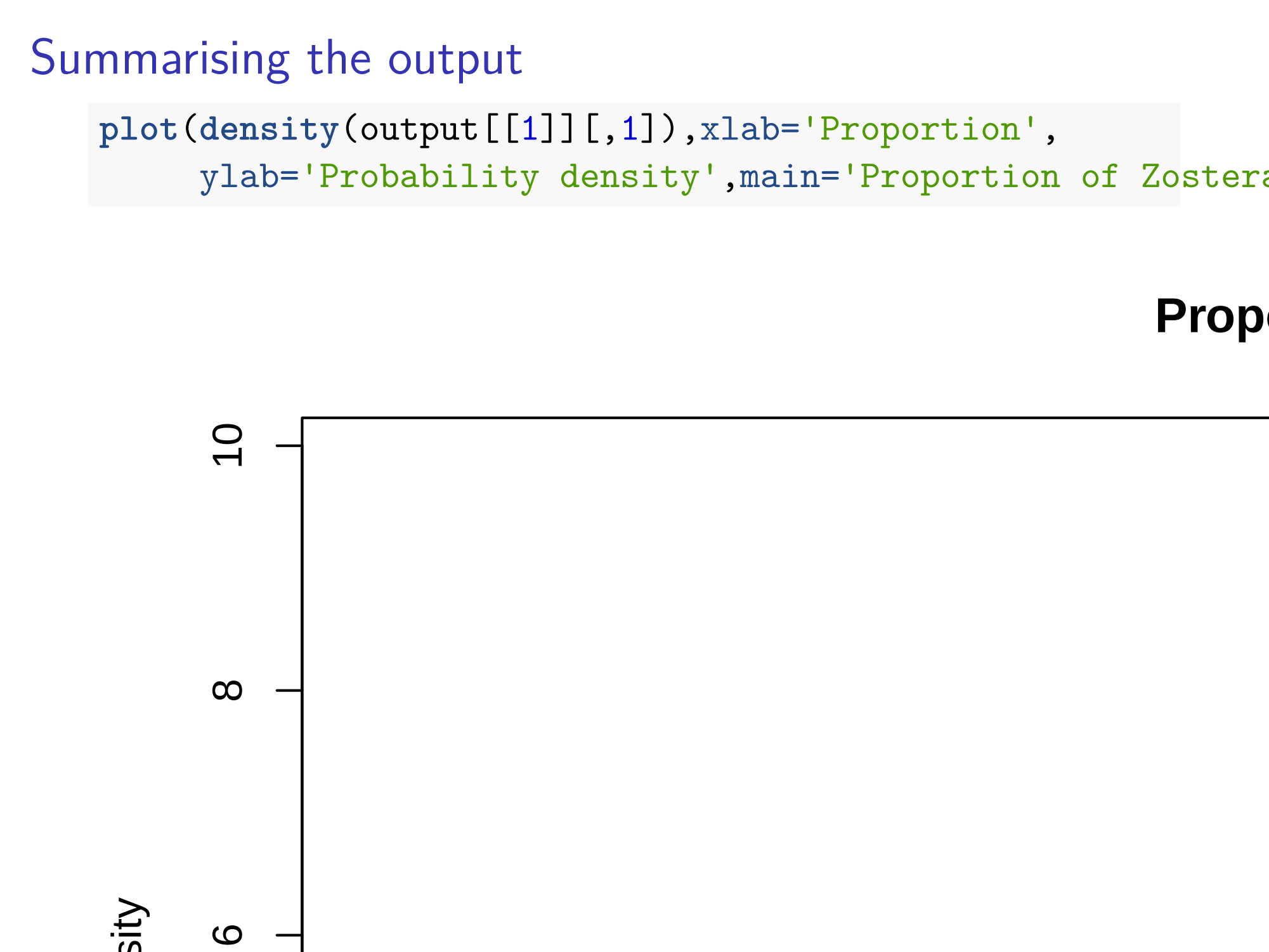
- The parameters for this simple model are s_1, s_2, p_1, p_2, and σ
- We have external data on the s_k values, so it makes sense to put a prior distribution $s_k \sim N(\mu_{s_k}, \sigma^2_{s_k})$ on each of these
- The dietary proportions must sum to 1, i.e. $p_2 = 1 - p_1$ so we have only have 1 parameter to place a prior on. We might use $p_1 \sim U(0, 1)$ if no prior knowledge.
- An alternative is the Beta distribution which can put more weight on lower or higher proportions
- We usually have little information on σ, but the isospace plot will usually give a rough guide to the likely range of values

A simple SIMM in JAGS

```
modelstring ='
model {
  for(i in 1:N) {
    y[i] ~ dnorm(p_1*s_1+p_2*s_2,sigma^-2)
  }
  p_1 ~ dunif(0,1)
  p_2 <- 1-p_1
  s_1 ~ dnorm(s_1_mean,s_1_sd^-2)
  s_2 ~ dnorm(s_2_mean,s_2_sd^-2)
  sigma ~ dunif(0,10)
}
'
data=list(y=consumers,s_1_mean=sources[1,1],s_1_sd=sources
          s_2_mean=sources[2,1],s_2_sd=sources[2,2],
          N=length(consumers))
model=jags.model(textConnection(modelstring), data=data)
output=coda.samples(model=model,variable.names=c("p_1","p_2
                    n.iter=10000)
```

Summarising the output

```
plot(density(output[[1]][,1]),xlab='Proportion',
     ylab='Probability density',main='Proportion of Zostera
```



Model checking and convergence

- How do you know whether the model fits the data well or not?
- How do you know that JAGS fitted the model OK?
- The model fit can be checked by running *cross-validation* (leaving out chunks of the data and getting the model to predict the y values of the left out data) or *posterior predictive* checks, amongst many other methods
- The fitting performance can be evaluated via *convergence checking*. This involves looking at the posterior samples and checking that the values are stable
- Another questions (which we will look at in a later session) is whether this is the 'best' model for the data

Model checking

- Adding a posterior predictive check is as simple as adding an extra line to the JAGS code

```
modelstring ='
  ...
  for(i in 1:N) {
    y[i] ~ dnorm(p_1*s_1+p_2*s_2,sigma^-2)
    y_pred[i] ~ dnorm(p_1*s_1+p_2*s_2,sigma^-2)
  }
  ...
'
```

- `y_pred` is included as another parameter, and is thus estimated as part of the model.
- We then have both the true y values and some estimated y values from the model

Model checking output

```
y_pred_quantiles = apply(output[[1]],2,'quantile',probs=c(0
round(cbind(data$y,t(y_pred_quantiles)),2)
```

```
##                          5%   95%
## y_pred[1] -11.36 -14.41 -8.99
## y_pred[2] -11.88 -14.40 -9.07
## y_pred[3] -10.60 -14.53 -9.02
## y_pred[4] -11.25 -14.48 -9.01
## y_pred[5] -11.66 -14.40 -9.01
## y_pred[6] -10.41 -14.47 -9.01
## y_pred[7] -10.88 -14.52 -9.06
## y_pred[8] -14.73 -14.55 -9.04
## y_pred[9] -11.52 -14.46 -9.04
```

Just 1 observation outside the 90% CI. Looks to be an OK model.

Convergence checking

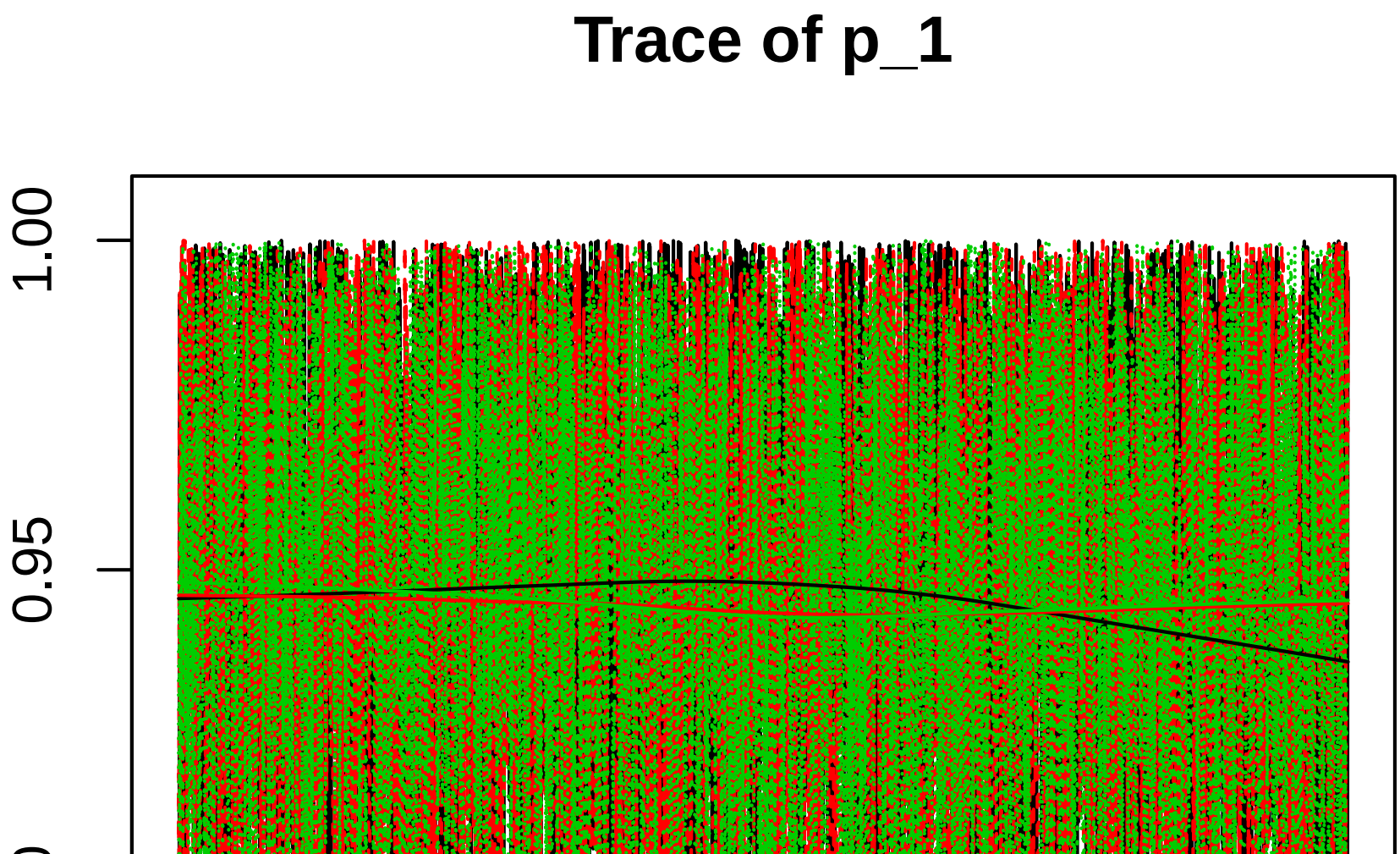
- When JAGS runs a model it creates initial guesses of the parameter values and then creates many consecutive samples moving away from the initial values towards the true posterior distribution
- Mathematical theory says that the samples must eventually come from the posterior distribution but this may take a very long time!
- Another method for ensuring convergence is to start JAGS with multiple different starting values and see if each model run (known as a *chain*) converges to the same posterior distribution
- You can supply initial guesses and the number of chains to JAGS when you run it:

```
inits = function() {
  list('p_1'=runif(1),'s_1'=rnorm(1),'s_2'=runif(1),
       'sigma'=runif(1,0,10))
}
model=jags.model(textConnection(modelstring), data=data, in
```

Convergence checking 2

- You can start by plotting the output from JAGS:

```
plot(output)
```



Convergence checking 3

- The different colours show the three chains. The location and variability should be broadly the same between chains
- There are some useful statistical tests for convergence, including the *Geweke* test which looks to see whether the mean is stable in each half of the iterations, or the *Brooks, Gelman, Rubin* (BGR) test which looks to see whether different chains match
- The Geweke test only requires one chain and is used by SIAR
- The BGR test is best if you have multiple chains

Convergence checking 4

- If you started by choosing bad initial values you might want to remove an initial chunk of the samples. This is known as the burn in and can be set using the `n.adapt` command in the `jags.model` function call
- Ideally the samples from the posterior distribution should be independent. If the algorithm isn't working well you can *thin* them out with the `thin` argument in the `coda.samples` function. The auto-correlation plot produced by `acf` in `R` can tell you whether you need to thin or not
- Finally, we need to choose the number of iterations. For very simple models 1,000 is usually fine, but for very complicated models you can sometimes need hundreds of thousands or millions. 10,000 is usually a good number for most problems. You can set the number of iterations in JAGS with `n.iter`

SIAR has its own commands for dealing with burn-in/thinning/iterations

Summary

- A SIMM is very similar to a linear regression. Things get slightly more complicated when we move to multiple isotopes
- The priors for a SIMM involve distributions for the source values, the dietary proportions, and the residual standard deviation
- When running a Bayesian model, remember to check your model (if possible) using posterior predictive checks, and convergence checking