Module 3: The JAGS software with simple examples

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

Learning outcomes

- Understand the code and the output from a JAGS model
- Be able to write and run JAGS models for linear and logistic regression
- Be able to change the prior distributions in a JAGS model
- Be comfortable with plotting and manipulating the output from JAGS

What is JAGS?

- JAGS stands for Just Another Gibbs Sampler and is written by Martyn Plummer
- It is a very popular tool for creating posterior distributions given a likelihood, prior, and data
- It uses a technique known as Gibbs sampling to produce samples of the parameters from the posterior distribution
- There are numerous implementations of JAGS. We will use the `R` package `R2jags`

Why JAGS?

There are several alternative ways of computing posterior distributions in `R` other than JAGS:

- Stan: this is a new-ish package which supposedly runs faster than JAGS and is slightly more flexible in the probability distributions it allows. However, it currently doesn't allow for discrete parameters and is painfully slow to compile each model
- WinBUGS/OpenBUGS: this was the first widely used software to compute posterior distributions. It's a little bit old now, only works on Windows, and is annoyingly point-and-click slow
- Lots of other R packages (see here). Most of these tend to only fit a certain class of models, or are platform specific
- If you find any other good ones, let me know!

The general framework for running jags

1. Write some model code in JAGS' own language
2. Collect your data together in a list
3. Tell JAGS which parameters you want it to keep
4. (Optional) Give JAGS a function or a list of starting values for the parameters
5. Call the `jags` function to run the model and store the results in an object
6. Take the posterior samples and manipulate them in any way you like

Model code

- Store the model code in a string
- Make sure it lists the likelihood and the priors clearly
- Use # for comment, <- for assignment, and ~ for distributions
- Refer back to the JAGS manual for more complicated functions

Example:

```
model_code = '
model {
    # Likelihood
    y ~ dnorm(theta, 1)
    # Prior
    theta ~ dnorm(0, 1)
}
'
```

Setting up the data and parameters

- Store the data in a *named list* so that JAGS can find the objects it needs
- JAGS will assume that anything you have not provided is a parameter
- The list can also include matrices and vectors

Example:

```
model_data = list(y = 1)
```

- Tell JAGS which parameters you want it to keep
- To save storage, JAGS will only output the parameters you tell it to, the posterior for any others is calculated but not stored

Example:

```
model_parameters = c('theta')
```

Step 3: running jags

```
library(R2jags)
model_run = jags(data = model_data,
                 parameters.to.save = model_parameters,
                 model.file = textConnection(model_code))
```

- The object `model_run` now stores the posterior distribution (and a load of other things)
- It's quite fiddly to get at the parameters but this is the usual route:

```
hist(model_run$BUGSoutput$sims.list$theta, breaks = 30)
```

Gibbs sampling in one slide

- Gibbs sampling is a trial and error algorithm. It starts with initial guesses of the parameters and then tries to find new ones which match the likelihood and prior better than the previous guesses
- Over thousands and thousands of iterations it should eventually converge towards the most likely parameter values
- A clever step in the algorithm means that it occasionally accepts worse guesses. This is what builds up a full posterior probability distribution rather than the most likely guesses
- The algorithm can fail if the starting values are bad, if it gets stuck in a local maximum, or if it guesses new parameter values that are too similar to the previous (meaning that it doesn't reach the most likely values)

Extra options when running JAGS

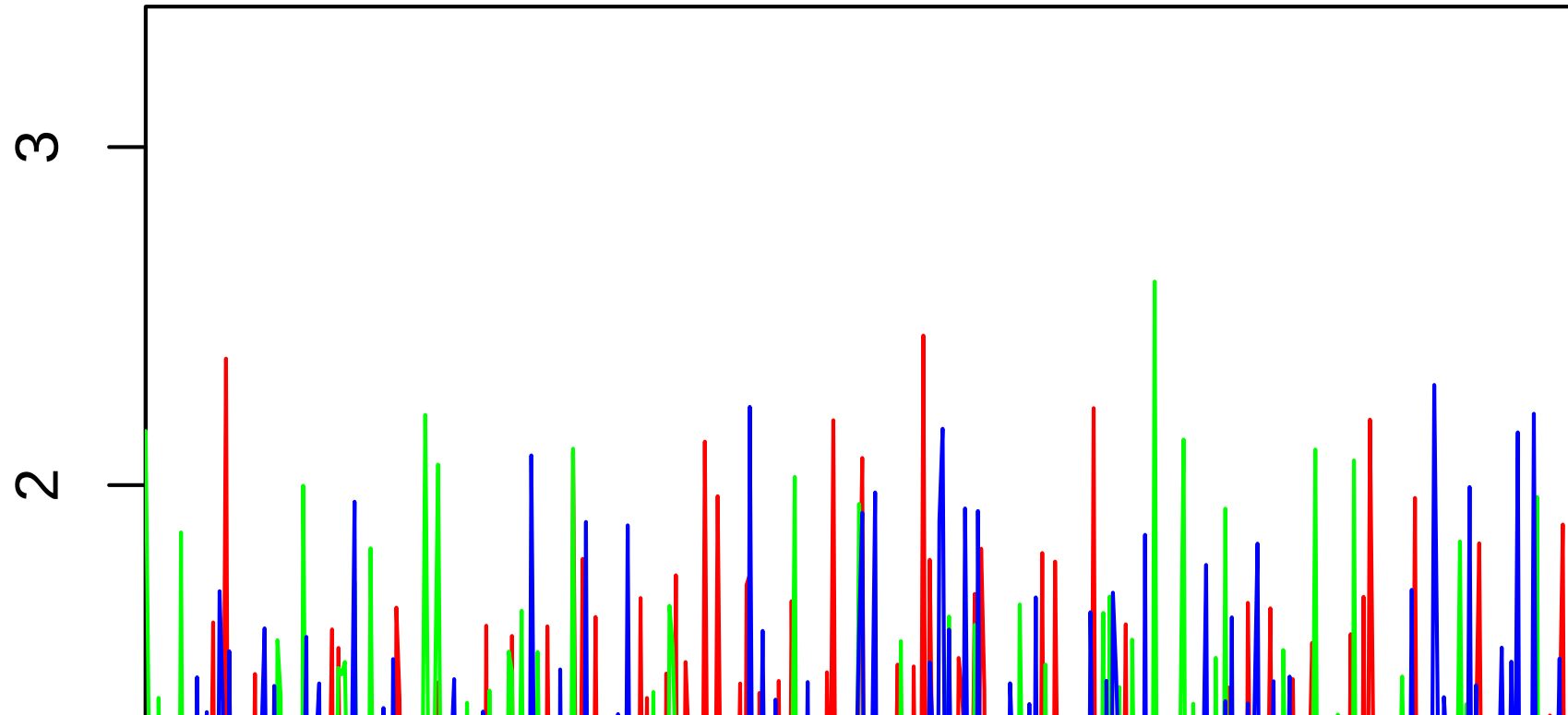
The usual way to run JAGS is to set some small additional options:

- We start the algorithm from multiple different starting values. This is known as running multiple *chains*
- We discard the first chunk of iterations whilst the algorithm is warming up. This is known as the *burn in* period
- We often only keep every ith iteration to avoid the problem of values being too similar. This is known as *thinning*

Checking convergence

If the algorithm has worked properly, we should see something like this for each parameter:

```
traceplot(model_run, varname = 'theta')
```



Checking convergence 2

Aside from the trace plot, the `R2jags` package also creates something called the Brooks-Gelman-Rubin (BGR) diagnostic which measures whether the mean/variance of each chain matches. It produces a value known as the R-hat value which should be close to 1 if we are sampling from the posterior distribution

```
print(model_run)
```

```
## Inference for Bugs model at "5", fit using jags,
##  3 chains, each with 2000 iterations (first 1000 discarc
##  n.sims = 3000 iterations saved
##          mu.vect sd.vect   2.5%   25%   50%   75% 97.5%
## theta      0.491   0.709 -0.898 0.016 0.493 0.965 1.887
## deviance   2.599   1.020  1.839 1.925 2.222 2.879 5.479
##
## For each parameter, n.eff is a crude measure of effectiv
## and Rhat is the potential scale reduction factor (at cor
##
## DIC info (using the rule, pD = var(deviance)/2)
```

Example 1. linear regression

Consider a simple linear regression model with one response variable y_t, $t = 1, \ldots, T$ and one explanatory variable x_t. The usual linear regression model is written as:

$$y_t = \alpha + \beta x_t + \epsilon_t;\ \epsilon_t \sim N(0, \sigma^2)$$

This can be written in likelihood format as:

$$y_t \sim N(\alpha + \beta x_t, \sigma^2)$$

so we have three parameters, the intercept α, the slope β, and the residual standard deviation σ. This latter parameter must be positive

In the absence of further information, we might use the priors $\alpha \sim N(0, 100^2)$, $\beta \sim N(0, 100^2)$, and $\sigma \sim U(0, 10)$

Example model code

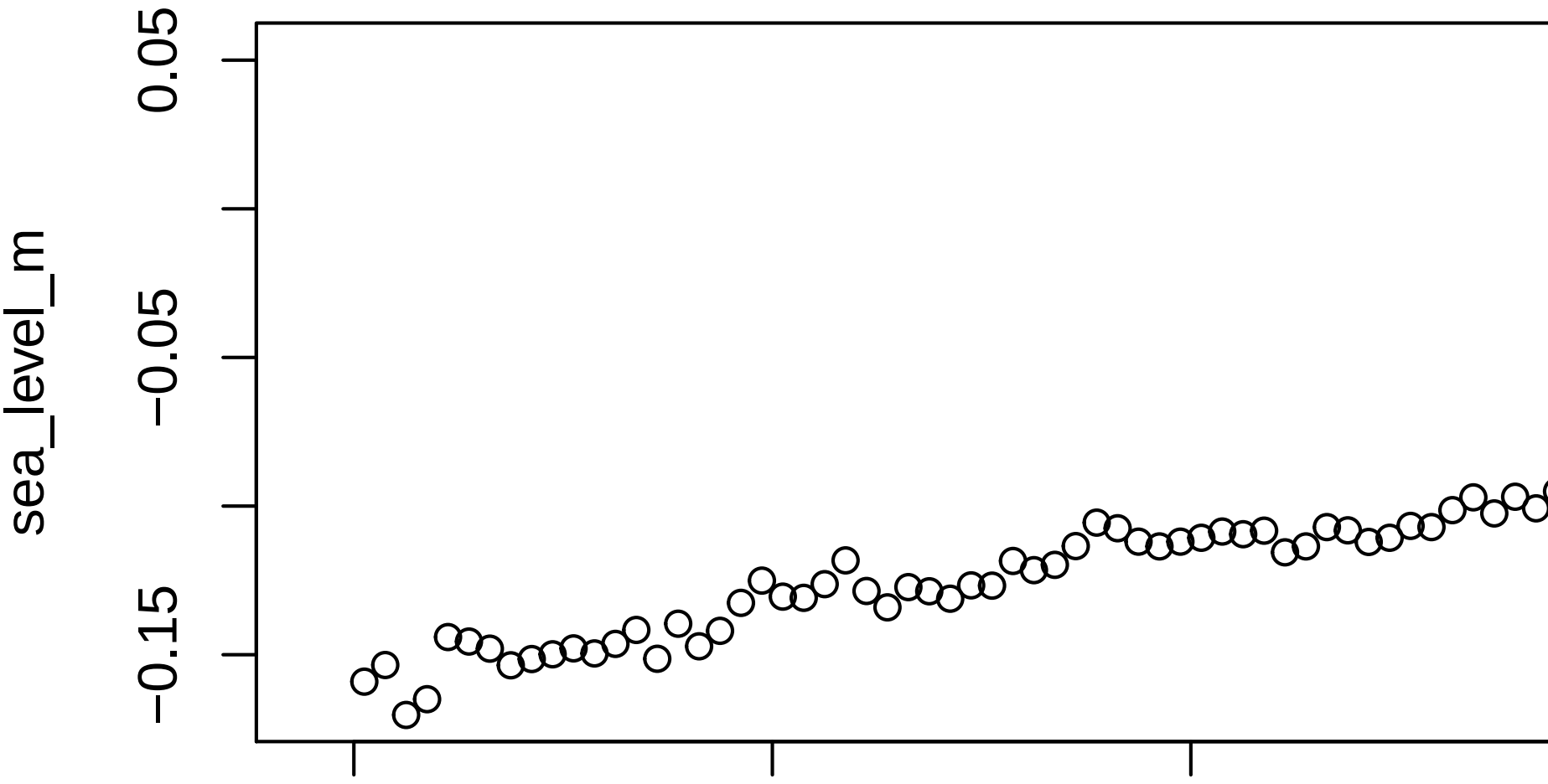
```
model_code = '
model
{
  # Likelihood
  for (t in 1:T) {
    y[t] ~ dnorm(alpha + beta * x[t], tau)
  }

  # Priors
  alpha ~ dnorm(0, 0.01) # = N(0,10^2)
  beta ~ dnorm(0, 0.01)
  tau <- 1/pow(sigma, 2) # Turn precision into standard dev
  sigma ~ dunif(0, 10)
}
'
```

Example data

See the file `jags_linear_regression.R` for more on this example

```
sea_level = read.csv('https://raw.githubusercontent.com/anc
with(sea_level,plot(year_AD,sea_level_m))
```



Create data list, parameters, and run

```
real_data = with(sea_level,
                 list(T = nrow(sea_level),
                      y = sea_level_m,
                      x = year_AD))

model_parameters =  c("alpha", "beta", "sigma")

real_data_run = jags(data = real_data,
                     parameters.to.save = model_parameters,
                     model.file=textConnection(model_code),
                     n.chains=4, # 4 chains
                     n.iter=1000, # 1000 iterations
                     n.burnin=200, # Discard first 200
                     n.thin=2) # Take every other iteration
```

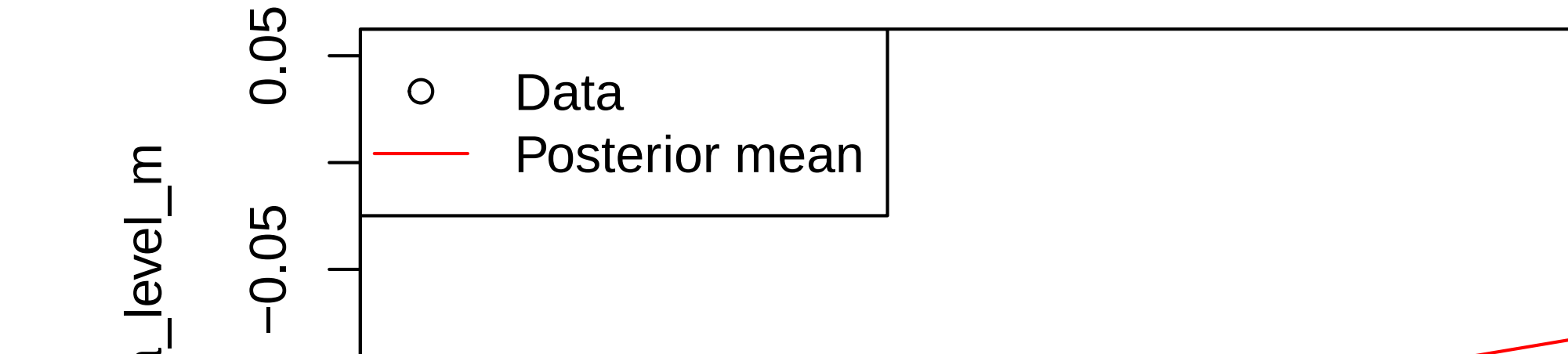
Check convergence

```
print(real_data_run)
```

```
## Inference for Bugs model at "6", fit using jags,
##  4 chains, each with 1000 iterations (first 200 discarde
##  n.sims = 1600 iterations saved
##           mu.vect sd.vect     2.5%      25%      50%
## alpha      -3.062   0.040   -3.140   -3.089   -3.061
## beta        0.002   0.000    0.001    0.002    0.002
## sigma       0.009   0.001    0.008    0.008    0.009
## deviance -864.729   2.543 -867.655 -866.597 -865.394 -86
##            Rhat n.eff
## alpha     1.002  1300
## beta      1.002  1400
## sigma     1.003   900
## deviance 1.001  1600
##
## For each parameter, n.eff is a crude measure of effectiv
## and Rhat is the potential scale reduction factor (at con
```

Creating plots

```
alpha_mean = mean(real_data_run$BUGSoutput$sims.list$alpha)
beta_mean = mean(real_data_run$BUGSoutput$sims.list$beta)
with(sea_level,plot(year_AD,sea_level_m))
with(sea_level,
     lines(year_AD, alpha_mean + beta_mean * year_AD, col =
legend('topleft',
       legend = c('Data', 'Posterior mean'),
       lty=c(-1,1),
       pch=c(1,-1),
       col=c('black','red'))
```



Checking the assumptions

Calculate residuals and create a QQ-plot

```
res = with(sea_level, sea_level_m - alpha_mean - beta_mean
qqnorm(res)
qqline(res)
```

Example 2. logistic regression

- Just like linear regression but this time the response is a count, usually restricted to have an upper value
- If the data are binary we write:

$$P(y_t = 1) = p_t; \text{logit}(p_t) = \alpha + \beta x$$

where $\text{logit}(p_t) = \log(p_t/(1 - p_t))$

- The logit function changes p_t from being restricted between 0 and 1, to having any value positive or negative
- The likelihood version of this is:

$$y_t \sim Bin(K, p_t); \text{logit}(p_t) = \alpha + \beta x$$

- We have two parameters α and β. You could consider p_t to be a set of parameters too but they are deterministically related to α and β

Example with two explanatory variables

- The example in `jags_logistic_regression.R` has two explanatory variables, so:

$$\text{logit}(p_t) = \alpha + \beta_1 x_{1t} + \beta_2 x_{2t}$$

- We will again use vague priors on these (now three) parameters:

$$\alpha \sim \mathit{normal}(0, 10^2), \mathit{beta}_1 \sim \mathit{normal}(0, 10^2), \mathit{beta}_2 \sim \mathit{normal}(0, 10^2)$$

- The data are adapted from Royla and Dorazio (Chapter 2), and concern the number of survivors of a set of moths based on sex (x_1) and the dose of an insecticide (x_2)
- There are $T = 12$ experiments, each with $K = 20$ moths

Model code

```
model_code = '
model
{
  # Likelihood
  for (t in 1:T) {
    y[t] ~ dbin(p[t], K)
    logit(p[t]) <- alpha + beta_1 * x_1[t] + beta_2 * x_2[t
  }

  # Priors
  alpha ~ dnorm(0.0,0.01)
  beta_1 ~ dnorm(0.0,0.01)
  beta_2 ~ dnorm(0.0,0.01)
}
'
```

Running the model

```
T = 12
K = 20
y = c(1,4,9,13,18,20, 0,2,6,10,12,16)
sex = c(rep('male',6), rep('female',6))
dose = rep(0:5, 2)
sexcode = as.integer(sex == 'male')

real_data = list(T = T, K = K, y = y, x_1 = sexcode, x_2 =

model_parameters =  c("alpha", "beta_1", "beta_2")

real_data_run = jags(data = real_data,
                     parameters.to.save = model_parameters
                     model.file = textConnection(model_code
                     n.chains = 4,
                     n.iter = 1000,
                     n.burnin = 200,
                     n.thin = 2)
```

Checking convergence

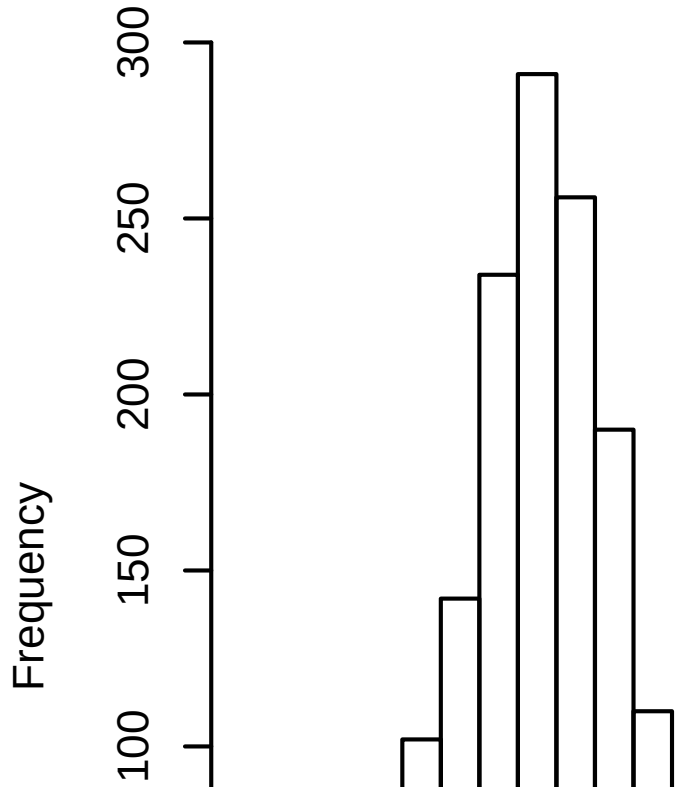
```
print(real_data_run)
```

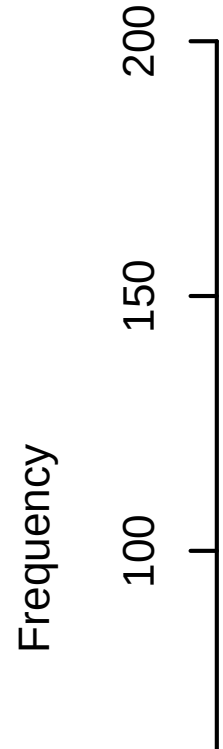
```
## Inference for Bugs model at "7", fit using jags,
##  4 chains, each with 1000 iterations (first 200 discarde
##  n.sims = 1600 iterations saved
##           mu.vect sd.vect   2.5%    25%    50%    75%  97
## alpha      -3.440   0.547 -4.426 -3.766 -3.460 -3.142 -2.
## beta_1      1.073   0.356  0.362  0.854  1.079  1.315  1.
## beta_2      1.062   0.159  0.763  0.973  1.065  1.156  1.
## deviance   40.508   5.961 37.085 38.136 39.417 41.281 47.
##
## For each parameter, n.eff is a crude measure of effectiv
## and Rhat is the potential scale reduction factor (at con
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 17.8 and DIC = 58.3
## DIC is an estimate of expected predictive error (lower d
```

Exploring the posterior

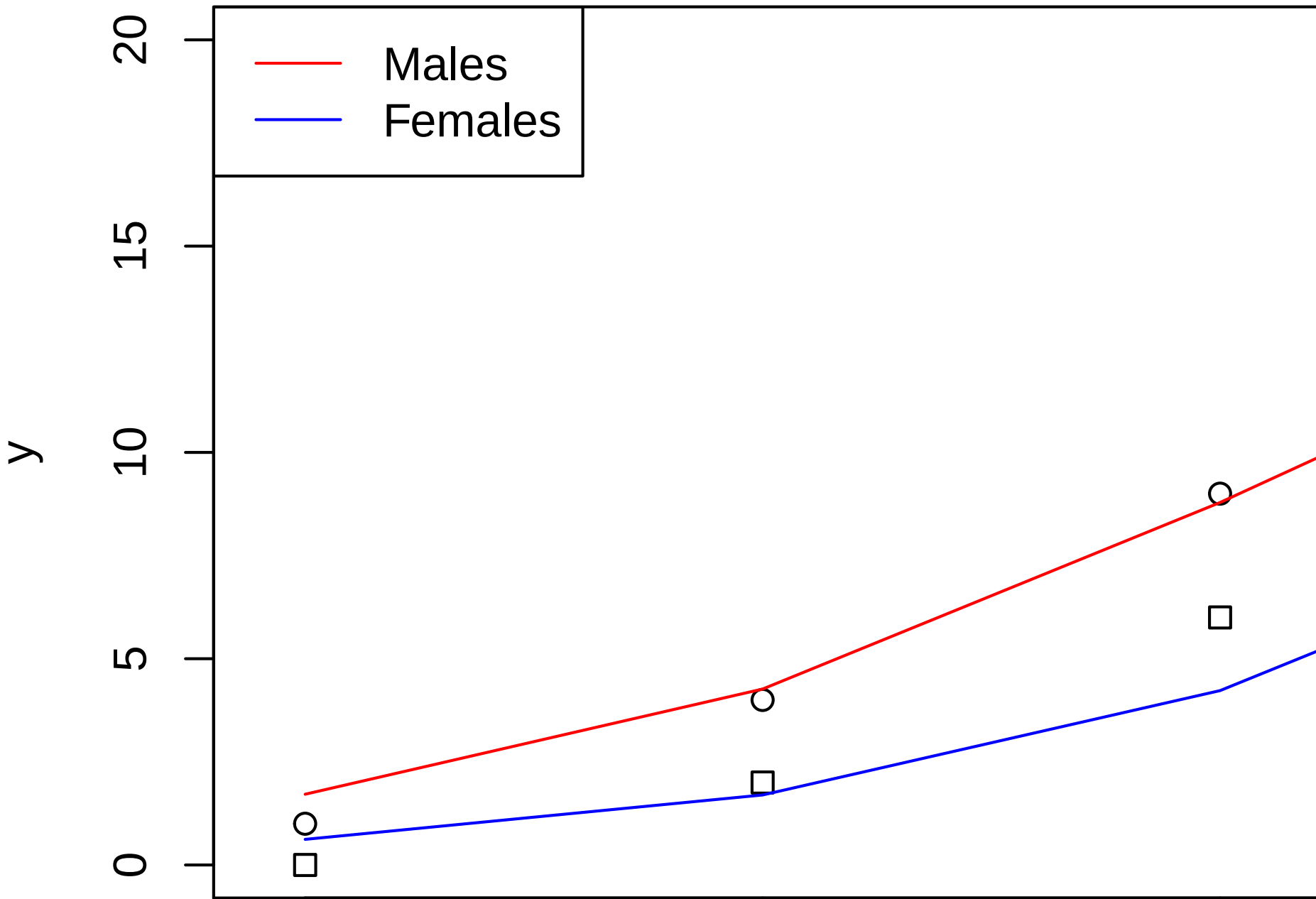
```
par(mfrow=c(1,3))
hist(real_data_run$BUGSoutput$sims.list$alpha, breaks = 30)
hist(real_data_run$BUGSoutput$sims.list$beta_1, breaks = 30
hist(real_data_run$BUGSoutput$sims.list$beta_2, breaks = 30
```

istogram of real_data_run$BUGSoutput$sims.list$stogram of real_data





Exploring the posterior 2



Summary and conclusions

- The code for a JAGS model follows its own special rules but always requires you to specify the likelihood and the priors
- We have seen how to create a Bayesian version of linear and logistic regression in JAGS
- Whenever we run a model we need to check convergence (using R-hat and trace plots)
- We can gain access to the posterior samples from JAGS and plot them any way we like
- There is much more detail in the `jags_linear_regression.R` and `jags_logistic_regression.R` files