Solutions for: Introduction to Bayesian Inference in WinBUGS 1.4.3 from R

These instructions are for completing the practicals using WinBUGS 1.4.3 from R.

All the data and other files you will need for the practicals were provided in a zip file, which you should have unzipped and saved in the directory
`C:‣bugscourse`

If you didn't download this zip archive, and/or don't have WinBUGS 1.4.3 installed on your laptop, please ask and we will be happy to help.

Solutions for all the exercises are provided in the directory `C:‣bugscourse‣solutions`

Installing and setting up WinBUGS 1.4.3 from R

Please just ask if you are unsure about installation or setup—we are happy to help!

Installing WinBUGS 1.4.3

WinBUGS 1.4.3 can be downloaded from:

`http://www.mrc-bsu.cam.ac.uk/software/bugs/the-bugs-project-winbugs/`

Installation instructions are provided on the above webpage.

Installing the R2WinBUGS R package

R2WinBUGS can be installed from within R by running

```
install.packages("R2WinBUGS")
```

Before every example

1. Open R
2. Attach the **R2WinBUGS** package

```
library(R2WinBUGS) # R2WinBUGS auto-loads coda, unlike R2OpenBUGS
## Loading required package:  coda
## Loading required package:  boot
```

3. Set the working directory to match the place you installed the BUGS model files and data, e.g.

```
setwd("C:/bugscourse/")
```

4. Load the bundled plotting functions provided with the course

```
# The following functions depend on the R package
# "denstrip". If necessary install using:
# install.packages("denstrip")
source("plot-functions.R")
```

5. Load the bundled data provided with the course

```
load(file="bugscourse.rda")
```

Practical 1: Introduction to Monte Carlo

A. Coins example

1. The model in `coins-model.txt` simulates throws of 10 balanced coins and records which give 8 or more heads. Open the model file and check you understand it—ask if you are unsure. You can either open the model file yourself by point-and-click, or open it from R using

```
file.show("coins-model.txt")
```

   The R script for this example is provided in:
   `bugscourse‣solutions‣practical1‣coins-winbugs-script.R`

2. The following **R2WinBUGS** command runs 10000 simulations from this model, recording the values of the parameters Y and P8.

```
coins.sim <- bugs(data="nodata.txt",
                  inits="nodata.txt",
                  model="coins-model.txt",
                  n.chains=1,
                  n.burnin=0,
                  n.iter=10000,
                  n.thin=1,
                  parameters.to.save=c("Y","P8"),
                  DIC=FALSE)
```

3. Typing the name of the object you created `coins.sim` prints summary statistics from the fitted model.

```
coins.sim
```

```
## Inference for Bugs model at "coins-model.txt", fit using WinBUGS,
##  1 chains, each with 10000 iterations (first 0 discarded)
##  n.sims = 10000 iterations saved
##    mean  sd 2.5% 25% 50% 75% 97.5%
## Y   5.0 1.6    2   4   5   6     8
## P8  0.1 0.2    0   0   0   0     1
```

   Some further notes on the `bugs` command in this case (which may become clearer after later lectures)

   - There is no data to load, so a dummy data file is provided in `nodata.txt`.
   - There is no need to provide initial values as this is a Monte Carlo forward sampling from a known distribution, and the dummy data file is also used as an (empty) initial values file.

- One parallel chain is run, with no burn-in iterations.
- DIC is not meaningful in this example, since we are not fitting a model to observed data.

The CODA package is then used for its MCMC output analysis facilities. To use it, we first convert the object returned by `bugs` to CODA format using the function `as.mcmc.list`.

```
coins.coda <- as.mcmc.list(coins.sim)
```

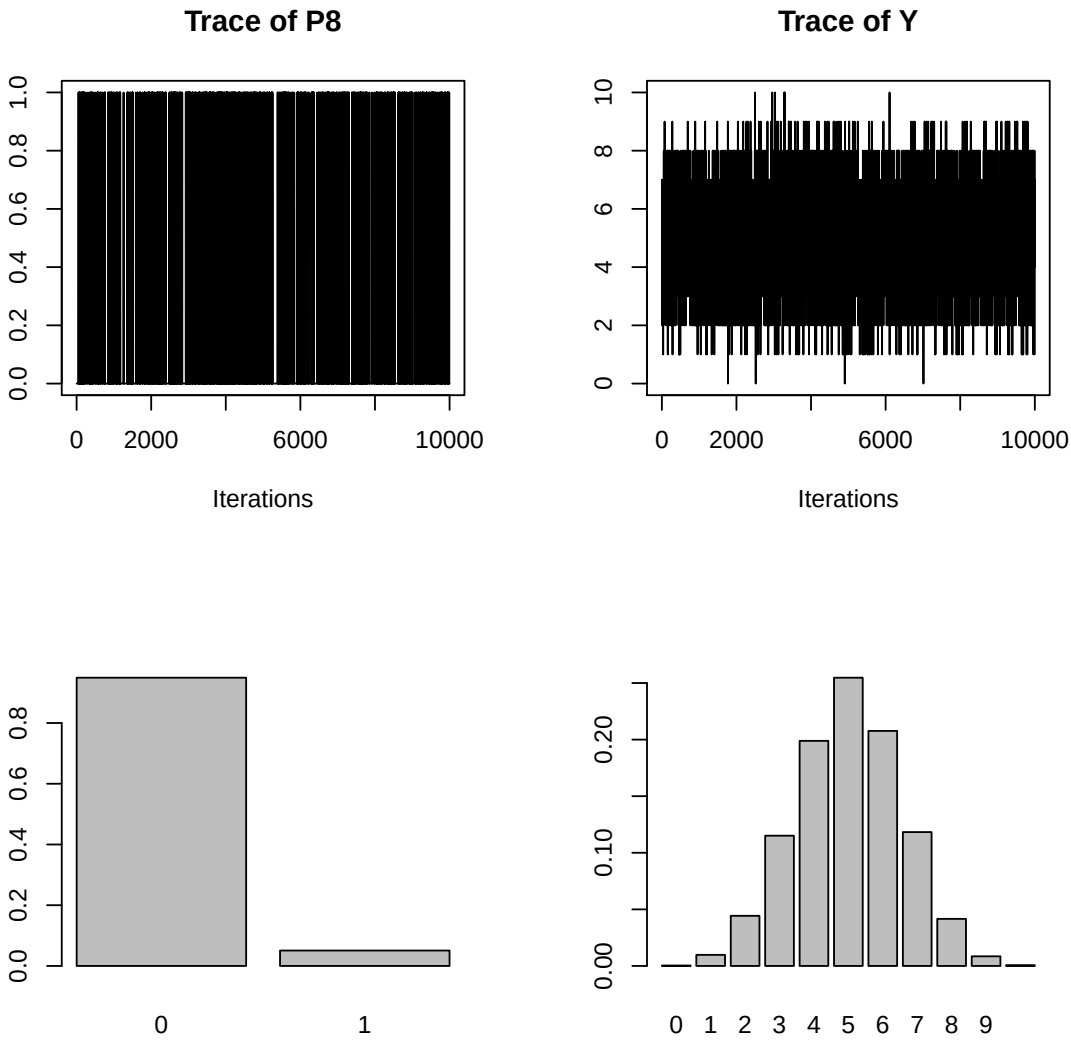
4. CODA has its own `summary()` function, which gives similar output to printing the object returned by `bugs()`, with the addition of the Monte Carlo standard error (labelled "Time-series SE" in the output).

```
summary(coins.coda)
```

```
##
## Iterations = 1:10000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##      Mean     SD Naive SE Time-series SE
## P8 0.0508 0.2196 0.002196       0.002209
## Y  5.0034 1.5601 0.015601       0.015601
##
## 2. Quantiles for each variable:
##
##    2.5% 25% 50% 75% 97.5%
## P8    0   0   0   0     1
## Y     2   4   5   6     8
```

We then arrange the next 4 plots in 2 rows and 2 columns, then produce the sample trace and posterior density estimates for the variables Y and P8.

```
par(mfrow=c(2,2))
traceplot(coins.coda)

# densplot.discrete() provided in plot-functions.R
# CODA's default densplot() is misleading for discrete parameters
densplot.discrete(coins.coda)
```



5. By editing the model file, find the probability that a clinical trial with 30 subjects, each with probability 0.7 of response, will show 15 or fewer responses.

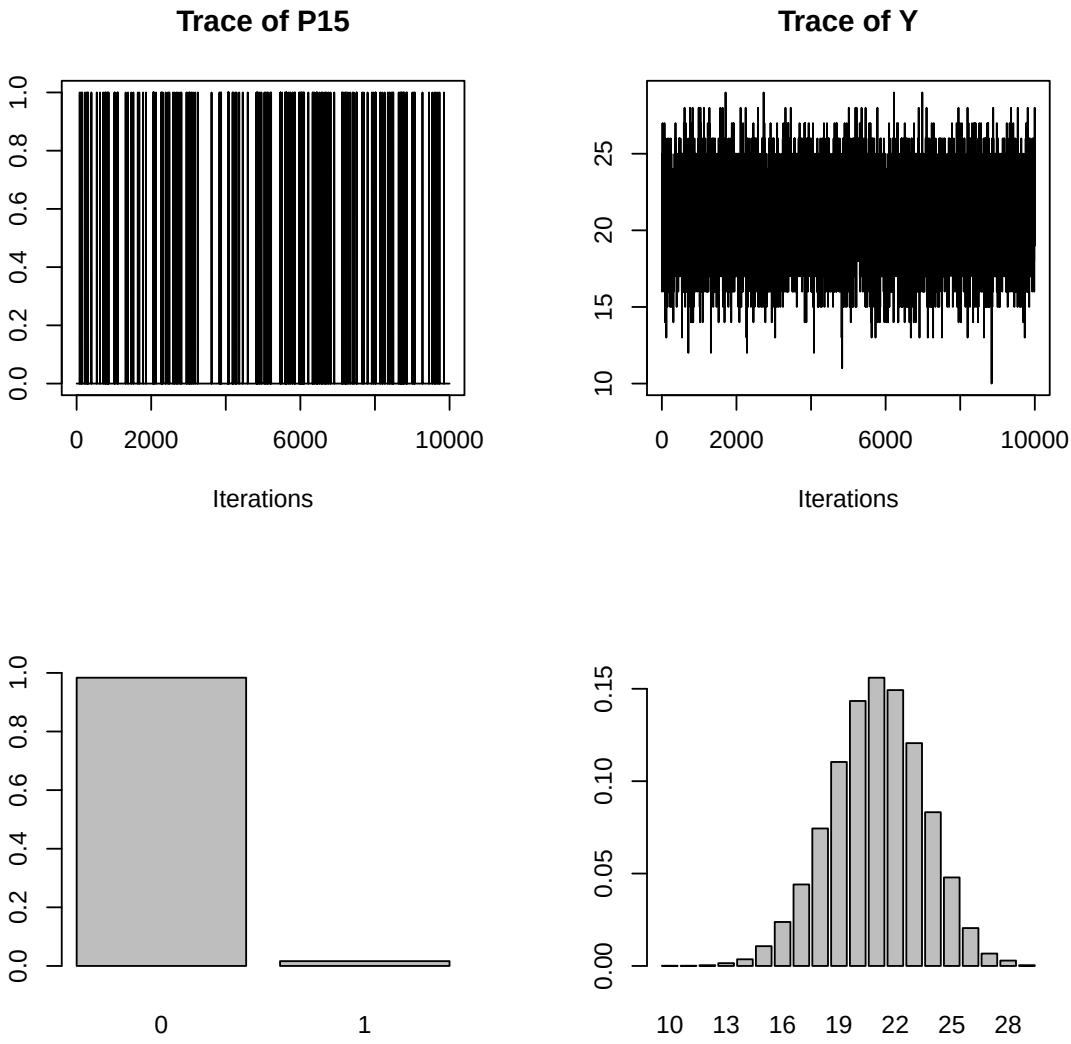
```
model <- normalizePath("solutions/practical1/coins-model-A5.txt")
coins.A5.sim <- bugs(data="nodata.txt",
                     inits="nodata.txt",
                     model=model,
                     n.chains=1,
                     n.burnin=0,
                     n.iter=10000,
                     n.thin=1,
                     parameters.to.save=c("Y","P15"),
                     DIC=FALSE)
```

```
coins.A5.coda <- as.mcmc.list(coins.A5.sim)
```

```
summary(coins.A5.coda)
```

```
##
## Iterations = 1:10000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##        Mean    SD Naive SE Time-series SE
## P15  0.0164 0.127  0.00127       0.001205
## Y   21.0080 2.517  0.02517       0.025172
##
## 2. Quantiles for each variable:
##
##     2.5% 25% 50% 75% 97.5%
## P15    0   0   0   0     0
## Y     16  19  21  23    26
```

```
par(mfrow=c(2, 2))
traceplot(coins.A5.coda)
densplot.discrete(coins.A5.coda)
```



So the chance is around 1.6% of getting 15 or fewer responses.

B. Drug example from lecture 1

1. Run the model code in `drug-MC-model.txt`, obtaining the results shown in the lectures (slides 1-24 and 1-25). You should be able to run it using the previous instructions (question A). If stuck, please ask!

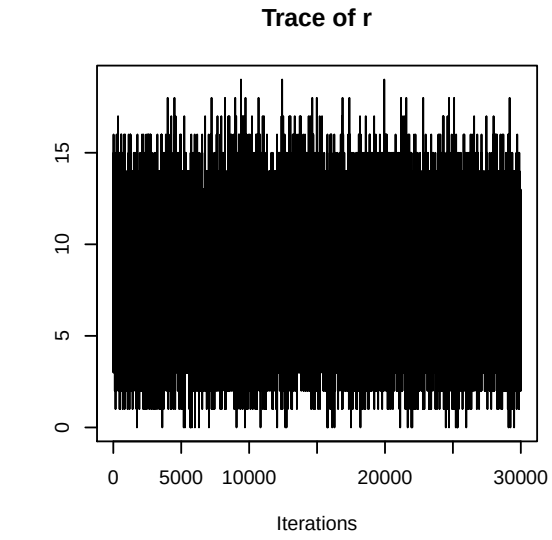
```
drug.sim <- bugs(model="drug-MC-model.txt",
                 data="nodata.txt",
                 inits="nodata.txt",
                 n.chains=1,
```

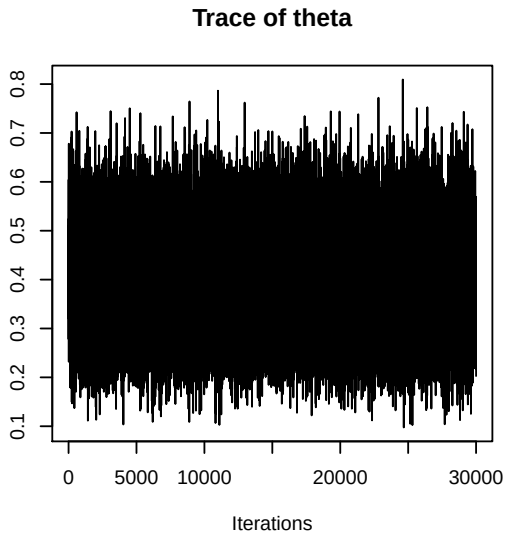
```
                  n.burnin=0,
                  n.iter=30000,
                  n.thin=1,
                  parameters.to.save=c("theta","r","P.crit"),
                  DIC=FALSE)
drug.sim
```

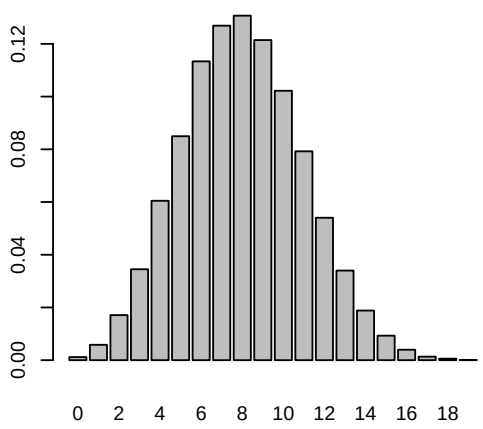
```
## Inference for Bugs model at "drug-MC-model.txt", fit using WinBUGS,
##  1 chains, each with 30000 iterations (first 0 discarded)
##  n.sims = 30000 iterations saved
##         mean  sd 2.5% 25% 50%  75% 97.5%
## theta   0.4 0.1  0.2 0.3 0.4  0.5   0.6
## r       8.0 2.9  3.0 6.0 8.0 10.0  14.0
## P.crit  0.0 0.1  0.0 0.0 0.0  0.0   0.0
```

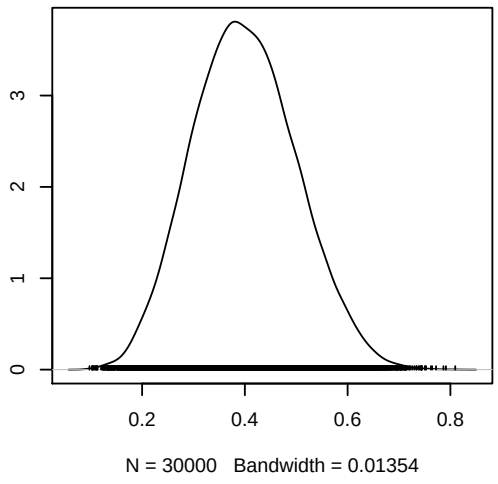
```
drug.coda <- as.mcmc.list(drug.sim)
```

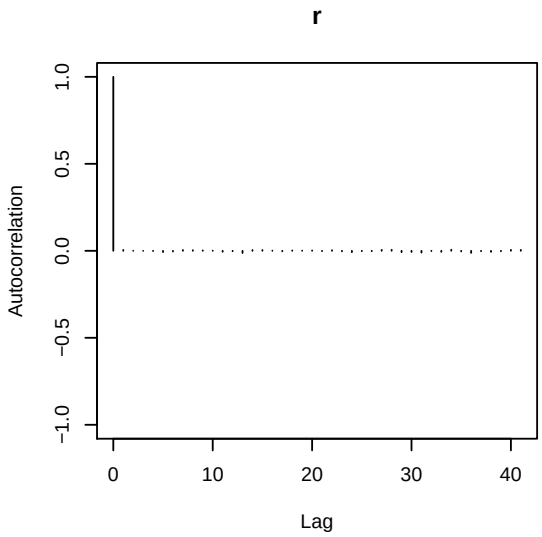
```
drug.coda <- drug.coda[,c("r","theta")]
par(mfrow=c(3,2))
traceplot(drug.coda)
densplot.discrete(drug.coda[, "r"]) # r is a discrete parameter
densplot(drug.coda[, "theta"]) # theta is a continuous parameter
autocorr.plot(drug.coda, auto.layout=FALSE)
```

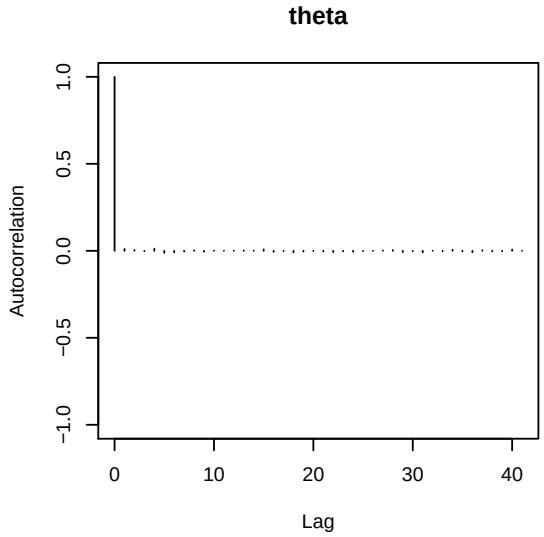

Trace of r
0
5
10
15
0
5000
10000
20000
30000
Iterations


Trace of theta
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
5000
10000
20000
30000
Iterations


0.00
0.04
0.08
0.12
0
2
4
6
8
10
12
14
16
18


0
1
2
3
0.2
0.4
0.6
0.8
N = 30000 Bandwidth = 0.01354


r
Autocorrelation
−1.0
−0.5
0.0
0.5
1.0
0
10
20
30
40
Lag


theta
Autocorrelation
−1.0
−0.5
0.0
0.5
1.0
0
10
20
30
40
Lag

2. Edit the model code to specify a Uniform(0, 1) prior on the response rate `theta`, and re-run the analysis. (Note: the syntax for the uniform prior in BUGS is `dunif(a, b)` where `a` and `b` are the lower and upper bounds. The values of `a` and `b` can either be specified in the data file, or directly in the BUGS code (e.g. `a <- 1`), or just replace `a` and `b` by their values in the `dunif` statement.)

   We change the prior by changing

   `theta ~ dbeta(9.2, 13.8)`

   to

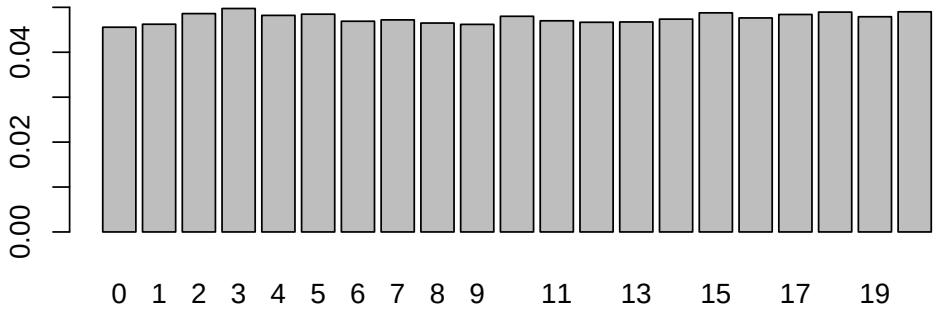
   `theta ~ dunif(0, 1)`

   in the model file.

```
model <- normalizePath("solutions/practical1/drug-MC-model-B2.txt")
drug.B2.sim <- bugs(data="nodata.txt",
                    inits="nodata.txt",
                    model=model,
                    n.chains=1,
                    n.burnin=0,
                    n.iter=30000,
                    n.thin=1,
                    parameters.to.save=c("theta","r","P.crit"),
                    DIC=FALSE)
drug.B2.coda <- as.mcmc.list(drug.B2.sim)
```

   - Plot the predictive distribution for the number of successes.

```
drug.B2.coda.r <- drug.B2.coda[[1]][,"r"]
densplot.discrete(drug.B2.coda.r)
```



- What is now the predictive probability that 15 or more patients will experience a positive response out of 20 new patients affected?

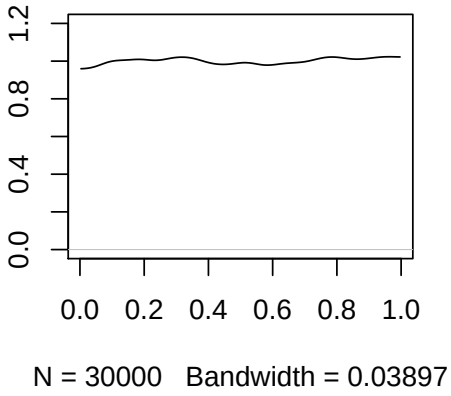
```
summary(drug.B2.coda)
```

```
##
## Iterations = 1:30000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 30000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean     SD Naive SE Time-series SE
## P.crit  0.2906 0.4541 0.002622       0.002653
## r      10.0424 6.0689 0.035039       0.035039
## theta   0.5021 0.2890 0.001668       0.001668
##
## 2. Quantiles for each variable:
##
##          2.5%    25%     50%     75%   97.5%
## P.crit 0.0000 0.0000  0.0000  1.0000  1.0000
## r      0.0000 5.0000 10.0000 15.0000 20.0000
## theta  0.0261 0.2523  0.5023  0.7544  0.9753
```

So about a 28.8% chance of more than 15. Note uniform predictive distribution on 0 to 20. Exact probability is therefore $6/21 = 0.286$.

Note we can check that the distribution of theta is now Uniform(0, 1), as required in the question:

```
drug.B2.coda.theta <- drug.B2.coda[,c("theta")]
densplot(drug.B2.coda.theta, ylim=c(0, 1.2), show.obs=FALSE)
```



C. Power example

1. Run the model given in `predpower-model.txt`. Check you get the answers shown in the lecture (slide 1-29).

```
predpower.sim <- bugs(model="predpower-model.txt",
                      data="nodata.txt",
                      inits="nodata.txt",
                      n.chains=1,
                      n.burnin=0,
                      n.iter=50000,
                      n.thin=1,
                      parameters.to.save=c("n","sigma","theta","power"),
                      DIC=FALSE)
predpower.sim
```

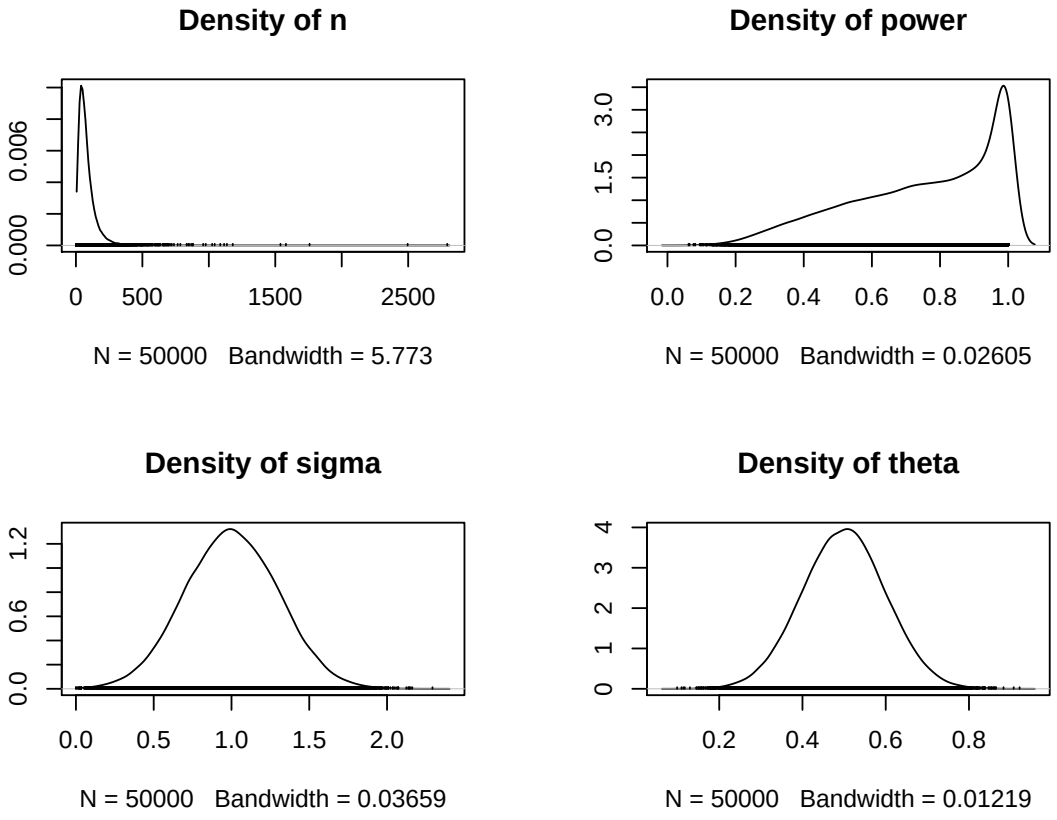
```
## Inference for Bugs model at "predpower-model.txt", fit using WinBUGS,
##  1 chains, each with 50000 iterations (first 0 discarded)
##  n.sims = 50000 iterations saved
##       mean   sd 2.5%  25%  50%   75% 97.5%
## n     79.4 68.6  9.8 37.5 63.2 101.0 245.7
## sigma  1.0  0.3  0.4  0.8  1.0   1.2   1.6
## theta  0.5  0.1  0.3  0.4  0.5   0.6   0.7
## power  0.8  0.2  0.3  0.6  0.8   1.0   1.0
```

```
predpower.coda <- as.mcmc.list(predpower.sim)
```

```
summary(predpower.coda)
```

```
##
## Iterations = 1:50000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 50000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##          Mean      SD  Naive SE Time-series SE
## n     79.4059 68.5502 0.3065658      0.3065658
## power  0.7554  0.2139 0.0009566      0.0009566
## sigma  1.0014  0.3005 0.0013438      0.0013591
## theta  0.4992  0.1001 0.0004478      0.0004478
##
## 2. Quantiles for each variable:
##
##         2.5%     25%     50%      75%   97.5%
## n     9.7893 37.4700 63.1700 101.0000 245.700
## power 0.2939  0.5993  0.7985   0.9526   1.000
## sigma 0.4135  0.7969  1.0000   1.2070   1.587
## theta 0.3026  0.4317  0.4998   0.5666   0.695
```

```
par(mfrow=c(2, 2))
densplot(predpower.coda)
```



2. What if the prior SD of sigma were reduced to 0.1?

   If prior SD of sigma is reduced to 0.1, replace the line for `prec.sigma` with line:

   ```
   prec.sigma <- 1/(0.1 * 0.1)    # transform sd to precision = 1/sd2
   ```

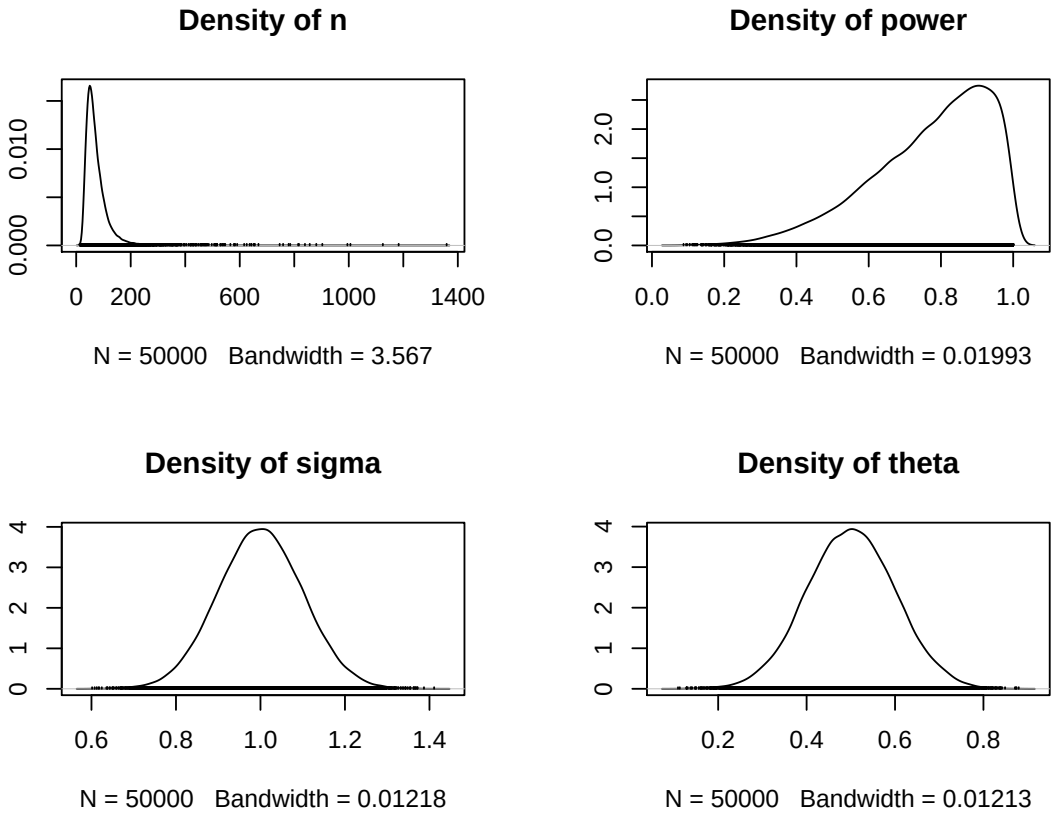
```
model <- normalizePath("solutions/practical1/predpower-model-C2.txt")
predpower.C2.sim <- bugs(data="nodata.txt",
                      inits="nodata.txt",
                      model=model,
                      n.chains=1,
                      n.burnin=0,
                      n.iter=50000,
                      n.thin=1,
                      parameters.to.save=c("n","sigma","theta","power"),
                      DIC=FALSE)
predpower.C2.coda <- as.mcmc.list(predpower.C2.sim)
```

```
summary(predpower.C2.coda)

##
## Iterations = 1:50000
```

```
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 50000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##          Mean       SD  Naive SE Time-series SE
## n     72.8874 44.59144 0.1994190      0.1994190
## power  0.7700  0.16372 0.0007322      0.0007322
## sigma  0.9999  0.10006 0.0004475      0.0004475
## theta  0.5006  0.09963 0.0004456      0.0004456
##
## 2. Quantiles for each variable:
##
##          2.5%     25%     50%     75%    97.5%
## n     28.4698 46.8800 62.3950 86.1400 177.9000
## power  0.3845  0.6681  0.8033  0.9008   0.9863
## sigma  0.8045  0.9324  0.9998  1.0680   1.1970
## theta  0.3046  0.4328  0.5006  0.5683   0.6966
```

So expected power goes up to about 77%, and 95% interval for sample size reduced to about 28 to 177.

```
par(mfrow=c(2, 2))
densplot(predpower.C2.coda)
```

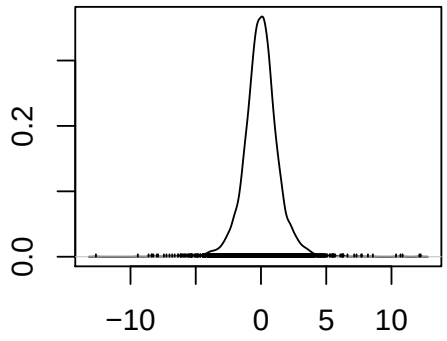


D. Writing your own code

1. Generate 10000 observations from a standard t distribution from 4 degrees of freedom [BUGS code, `y ~ dt(0,1,4)`], and plot the density. Would you consider this a heavy-tailed distribution compared to the normal?

```
model <- normalizePath("solutions/practical1/t-model-D1.txt")
t.D1.sim <- bugs(data="nodata.txt",
                 inits="nodata.txt",
                 model=model,
                 n.chains=1,
                 n.burnin=0,
                 n.iter=10000,
                 n.thin=1,
                 parameters.to.save="y",
                 DIC=FALSE)
t.D1.coda <- as.mcmc.list(t.D1.sim)
```

```
summary(t.D1.coda)

##
## Iterations = 1:10000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean             SD       Naive SE Time-series SE
##      1.653e-05      1.375e+00      1.375e-02      1.353e-02
##
## 2. Quantiles for each variable:
##
##       2.5%        25%        50%        75%      97.5%
## -2.6771750 -0.7290250  0.0001672  0.7187250  2.7542500

densplot(t.D1.coda)
```



N = 10000 Bandwidth = 0.1815

The true SD is $\sqrt{2} = 1.414$

2. Find by simulation the expectation of the cube of a normal random variable with mean 1 and standard deviation 2 (remember the BUGS parameterisation of the normal is in terms of the precision $= 1/\text{variance}$), and y^3 is written as `pow(y,3)`.

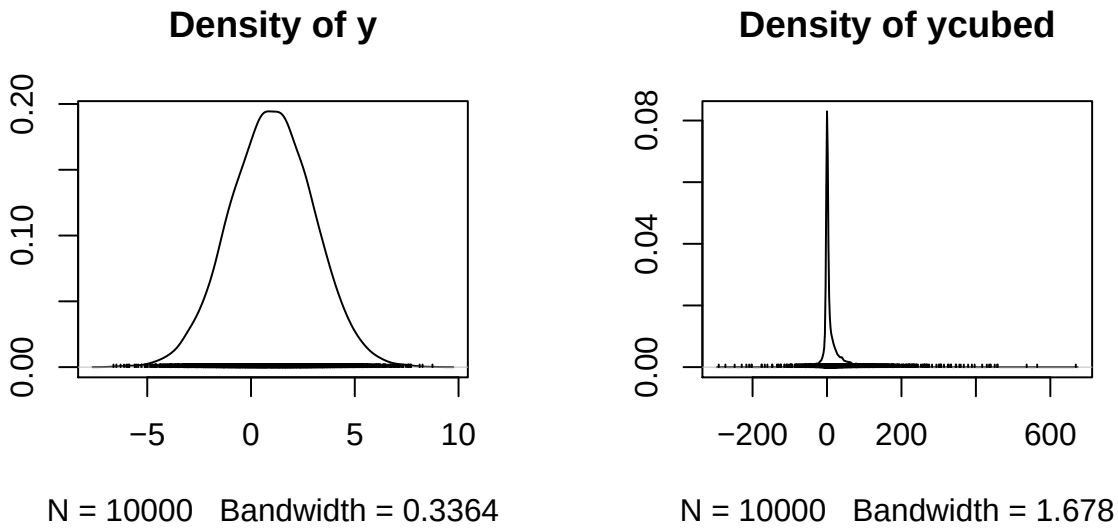
```
model <- normalizePath("solutions/practical1/cubed-model-D2.txt")
cubed.D2.sim <- bugs(data="nodata.txt",
```

```
                    inits="nodata.txt",
                    model=model,
                    n.chains=1,
                    n.burnin=0,
                    n.iter=10000,
                    n.thin=1,
                    parameters.to.save=c("y","ycubed"),
                    DIC=FALSE)
cubed.D2.coda <- as.mcmc.list(cubed.D2.sim)
```

```
summary(cubed.D2.coda)
```

```
##
## Iterations = 1:10000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##          Mean     SD Naive SE Time-series SE
## y       1.005  2.003  0.02003        0.01973
## ycubed 13.033 40.583  0.40583        0.40583
##
## 2. Quantiles for each variable:
##
##           2.5%     25%   50%    75%   97.5%
## y       -2.947 -0.3479 1.011  2.372   4.921
## ycubed -25.591 -0.0421 1.036 13.342 119.107
```

```
par(mfrow=c(1,2))
densplot(cubed.D2.coda)
```



The correct answer is 13: ycubed has very long tails!

Practical 2: Conjugate Bayesian inference

A. Drug example from lecture 2

The BUGS code for fitting the beta-binomial model (slide 2-20) to the drug data can be found in file `drug-model.txt`. The data are in the R list object `drug` and one set of initial values is given in `drug_in1`.

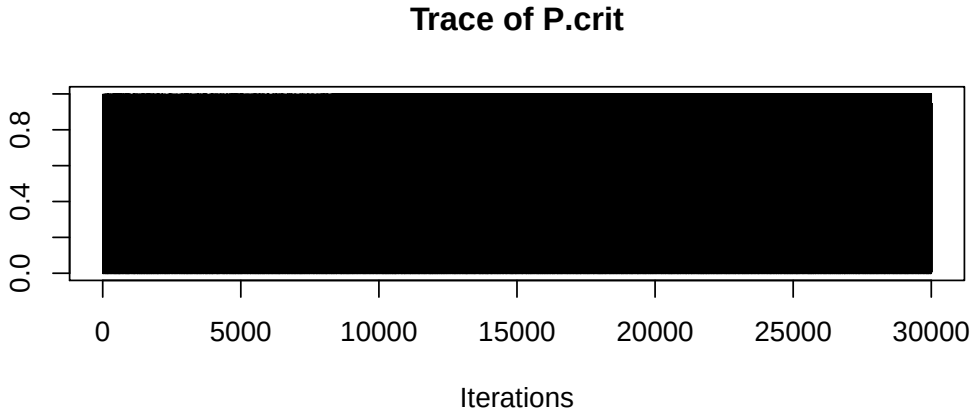
1. Carry out a WinBUGS run for this model. The data and initial values should be provided to the `bugs` function using the `data` and `inits` arguments respectively.

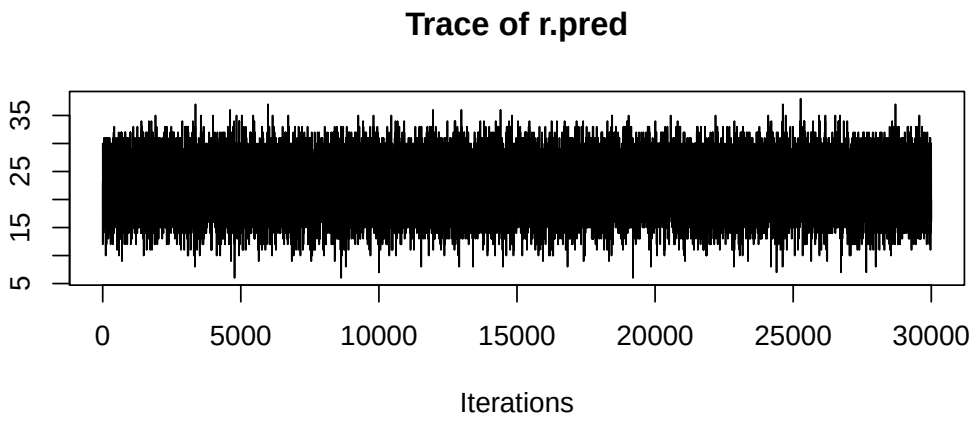
   (a) Make sure that you monitor the following parameters: the drug response rate, `theta`, the predicted number of positive responses in 40 future patients, `r.pred`, and the indicator of whether 25 or more positive responses are predicted, `P.crit`.

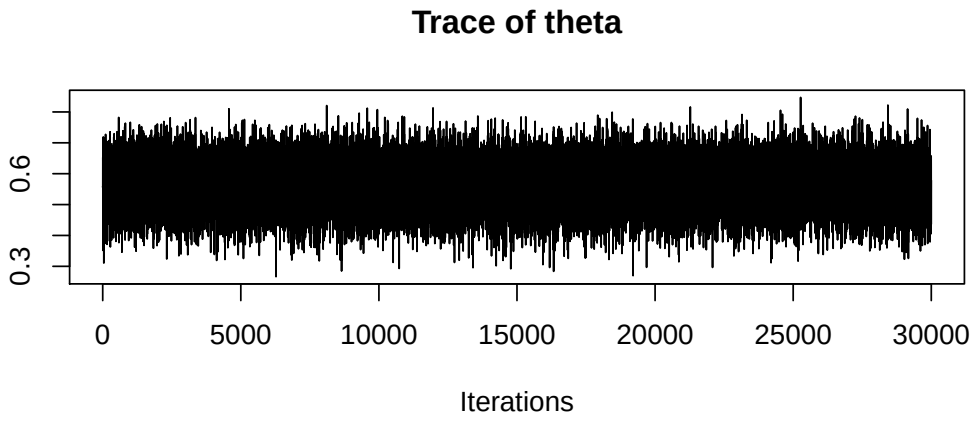
   (b) Run 30000 iterations (or more or fewer if you wish) to obtain samples from the posterior distribution of the model parameters.

```
parameters.to.save <- c("theta","r.pred","P.crit")
drug.A1.sim <- bugs(model="drug-model.txt",
                    data=drug,
                    inits=drug_in1,
                    n.chains=1,
                    n.burnin=0,
                    n.iter=30000,
                    n.thin=1,
                    parameters.to.save=parameters.to.save,
                    DIC=FALSE)
drug.A1.coda <- as.mcmc.list(drug.A1.sim)
```

```
traceplot(drug.A1.coda)
```







(c) Produce summary statistics for all the variables you have monitored.

- Provide a point and interval estimate for the treatment response rate
- What is the probability that 25 or more patients will experience a positive response out of 40 new patients to be administered the drug?

```
summary(drug.A1.coda)

##
## Iterations = 1:30000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 30000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
```

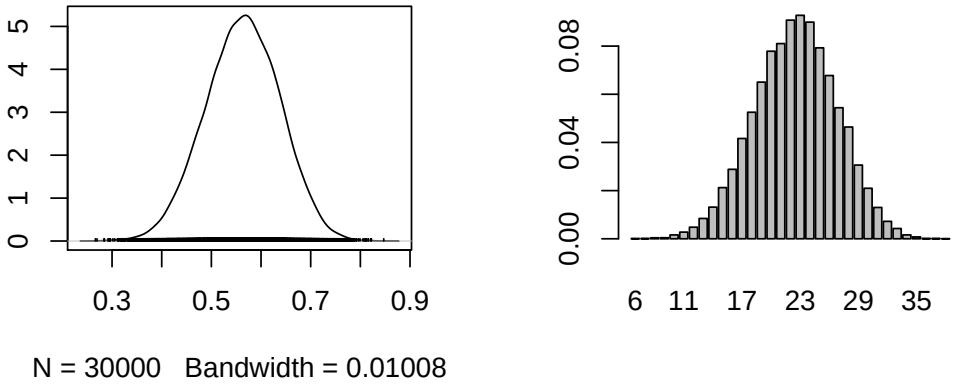
```
##           Mean      SD  Naive SE Time-series SE
## P.crit  0.3266 0.46897 0.0027076      0.0026711
## r.pred 22.5184 4.27604 0.0246877      0.0243453
## theta   0.5630 0.07475 0.0004315      0.0004315
##
## 2. Quantiles for each variable:
##
##           2.5%     25%     50%    75%   97.5%
## P.crit  0.0000  0.0000  0.0000  1.000  1.0000
## r.pred 14.0000 20.0000 23.0000 25.000 31.0000
## theta   0.4144  0.5124  0.5642  0.615  0.7054
```

So response rate is estimated to be 0.56 (95% interval 0.41 to 0.71). Compare these with the exact values, which are 0.56279 (2.5%ile = 0.41421, 97.5%ile = 0.70587).

Probability that at least 25 out of 40 new patients will respond is 0.33. The exact value (calculated analytically) is 0.32901.

(d) Produce kernel density plots of the posterior distribution for `theta` and the predictive distribution for `r.pred`

```
par(mfrow=c(1,2))
densplot(drug.A1.coda[, "theta"])
densplot.discrete(drug.A1.coda[, "r.pred"])
```



2. Compare the model code (and data) with the code for the Monte Carlo analysis of the drug data that you carried out in practical 1. Make sure you understand the difference between the two analyses.

The model specification is essentially the same in both cases (note that differences are just down to syntax, and whether or not fixed constants such as the parameter values of

the beta prior are specified directly in the BUGS code or in a separate data file; similarly choice of variable names makes no difference). The key difference that distinguishes the two models is that in the Monte Carlo analysis in practical 1, there is no observed data on `r` (the number of successes in 20 trials) and hence no learning about the success rate `theta`. Hence WinBUGS is generating samples of theta from the specified prior distribution, $Beta(9.2, 13.8)$. In practical 2, we have observed data on the number of successes; hence WinBUGS recognises that it needs to do posterior updating rather than just forward sampling from the prior, and will actually be generating samples of theta from its posterior distribution, not the prior.

3. Edit the model code to specify a Uniform(0, 1) prior on the response rate `theta` (or equivalently, a Beta(1, 1) prior), and re-run the analysis.

   Easiest way to specify this model is to edit the values of a and b in the data file to both be equal to 1 (recall that Beta(1, 1) is equivalent to Uniform(0, 1)).

```
# change a and b to 1 in the data
drug_data_A3 <- drug
drug_data_A3$a <- 1
drug_data_A3$b <- 1

drug.A3.sim <- bugs(model="drug-model.txt",
                    data=drug_data_A3,
                    inits=drug_in1,
                    n.chains=1,
                    n.burnin=0,
                    n.iter=30000,
                    n.thin=1,
                    parameters.to.save=c("theta","r.pred","P.crit"),
                    DIC=FALSE)
drug.A3.coda <- as.mcmc.list(drug.A3.sim)
```

   Alternatively, you could edit the model code and data file as follows:

```
model {
theta ~ dunif(0, 1)    # prior distribution
r ~ dbin(theta, n)    # sampling distribution
r.pred ~ dbin(theta, m)    # predictive distribution
P.crit<- step(r.pred - ncrit + 0.5)    # =1 if r.pred >= ncrit, 0 otherwise
}
```

   However, note that in WinBUGS all variables in the data file must be used in the model, so we must create an tweaked data file with `a` and `b` removed. i.e.

```
list(
r = 15,      # number of successes
```

```
n = 20,      # number of trials
m = 40,      # future number of trials
ncrit = 25   # critical value of future successes
)
```

```
## $r
## [1] 15
##
## $n
## [1] 20
##
## $m
## [1] 40
##
## $ncrit
## [1] 25
```

- How is the posterior estimate of `theta` affected?

```
summary(drug.A3.coda)
```

```
##
## Iterations = 1:30000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 30000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean      SD  Naive SE Time-series SE
## P.crit  0.8388 0.36769 0.0021229      0.0021229
## r.pred 29.0888 4.59619 0.0265361      0.0265361
## theta   0.7277 0.09244 0.0005337      0.0005337
##
## 2. Quantiles for each variable:
##
##           2.5%     25%     50%     75%   97.5%
## P.crit  0.0000  1.0000  1.0000  1.0000  1.0000
## r.pred 19.0000 26.0000 29.0000 32.0000 37.0000
## theta   0.5326  0.6671  0.7353  0.7953  0.8865
```

With Uniform prior, response rate is estimated to be 0.73 (95% interval 0.53 to 0.89). Compare with exact values of 0.72727 (2.5%ile = 0.52828, 97.5%ile = 0.88721)

- How is the probability that 25 or more patients will experience a positive response out

of 40 new patients affected?

Probability that at least 25 out of 40 new patients will respond is 0.84. Compare with exact value of 0.83645

B. THM data from Lecture 2

From scratch, implement the THM example of inference on the mean of a Normal distribution, slides 2-30 to 2-34. (Note—you will need to specify values for both the THM observations, rather than just inputting their mean as the data, so just enter both values as 130).

1. First implement the model with the informative prior specified in the lecture notes (slide 2-31). The syntax for the normal distribution in BUGS is `dnorm(mu, tau)` where `mu` is the mean and `tau` is the *precision* (= 1/variance)).

```
thm_model_B1_file <- normalizePath("solutions/practical2/thm-model-B1.txt")
thm_in1_B1 <- list(theta = 100)

# No data is needed here because it is included in the model
thm.B1.sim <- bugs(model=thm_model_B1_file,
                   data="nodata.txt",
                   inits=thm_in1_B1,
                   n.chains=1,
                   n.burnin=0,
                   n.iter=10000,
                   n.thin=1,
                   parameters.to.save=c("ypred","pmean.130","py.130","theta"),
                   DIC=FALSE)
thm.B1.coda <- as.mcmc.list(thm.B1.sim)
```

```
summary(thm.B1.coda)
```

```
##
## Iterations = 1:10000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                Mean     SD Naive SE Time-series SE
## pmean.130    0.3723 0.4834 0.004834       0.004925
## py.130       0.4274 0.4947 0.004947       0.004947
## theta      128.8928 3.3084 0.033084       0.032525
```

```
## ypred     128.9161 5.9644 0.059644       0.059644
##
## 2. Quantiles for each variable:
##
##             2.5%   25%   50%   75% 97.5%
## pmean.130    0.0   0.0   0.0   1.0   1.0
## py.130       0.0   0.0   0.0   1.0   1.0
## theta      122.4 126.6 128.9 131.2 135.4
## ypred      117.3 124.9 128.9 132.9 140.5
```

So posterior mean and 95% interval for mean THM level, theta, are 128.9 (122.4, 135.3), which essentially agrees with the exact answers in the lecture notes.

2. Now try using a non-informative prior on the unknown mean THM concentration (assume either a uniform prior with a wide range, or a normal prior with a large variance (small precision))

```
thm_model_B2a_file <- normalizePath("solutions/practical2/thm-model-B2a.txt")
thm_in1_B1 <- list(theta = 100)

# No data is needed here because it is included in the model
thm.B2a.sim <- bugs(model=thm_model_B2a_file,
                    data="nodata.txt",
                    inits=thm_in1_B1,
                    n.chains=1,
                    n.burnin=0,
                    n.iter=10000,
                    n.thin=1,
                    parameters.to.save=c("ypred","pmean.130","py.130","theta"),
                    DIC=FALSE)
thm.B2a.coda <- as.mcmc.list(thm.B2a.sim)
```

```
summary(thm.B2a.coda)
```

```
##
## Iterations = 1:10000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##               Mean     SD Naive SE Time-series SE
## pmean.130   0.4901 0.4999 0.004999       0.004999
```

```
## py.130      0.4921 0.5000 0.005000        0.005108
## theta     129.9695 3.5174 0.035174        0.035677
## ypred     129.9282 6.0938 0.060938        0.063389
##
## 2. Quantiles for each variable:
##
##             2.5%   25%   50%   75% 97.5%
## pmean.130    0.0   0.0   0.0   1.0   1.0
## py.130       0.0   0.0   0.0   1.0   1.0
## theta      123.1 127.6 129.9 132.4 136.8
## ypred      117.9 125.8 129.9 134.1 141.7
```

Under this model (uniform prior), the posterior mean and 95% interval for mean THM level, theta, are 130.0 (123.1, 136.8), which is slightly higher than under the informative prior (which was concentrated around 128).

```
thm_model_B2b_file <- normalizePath("solutions/practical2/thm-model-B2b.txt")
thm_in1_B1 <- list(theta = 100)

# No data is needed here because it is included in the model
thm.B2b.sim <- bugs(model=thm_model_B2b_file,
                    data="nodata.txt",
                    inits=thm_in1_B1,
                    n.chains=1,
                    n.burnin=0,
                    n.iter=10000,
                    n.thin=1,
                    parameters.to.save=c("ypred","pmean.130","py.130","theta"),
                    DIC=FALSE)
thm.B2b.coda <- as.mcmc.list(thm.B2b.sim)
```

```
summary(thm.B2b.coda)
```

```
##
## Iterations = 1:10000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##               Mean    SD Naive SE Time-series SE
## pmean.130   0.4993 0.500  0.00500        0.00500
## py.130      0.5029 0.500  0.00500        0.00500
```

```
## theta     130.0029 3.509  0.03509         0.03449
## ypred     130.0252 6.077  0.06077         0.06077
##
## 2. Quantiles for each variable:
##
##             2.5%   25% 50%   75% 97.5%
## pmean.130    0.0   0.0   0   1.0   1.0
## py.130       0.0   0.0   1   1.0   1.0
## theta      123.1 127.6 130 132.4 136.9
## ypred      118.2 125.9 130 134.1 141.9
```

Under this model (normal prior), the posterior mean and 95% interval for mean THM level, theta, are 130.0 (123.1, 136.8), which is slightly higher than under the informative prior (which was concentrated around 128).

3. For each model, include statements in your BUGS code to:

   (a) obtain a sample from the predictive distribution of a future THM concentration in the zone

   This can be done by adding the following line to the model:

   ```
   ypred ~ dnorm(theta, tau)
   ```

   (b) calculate the probability that the zone *mean* THM concentration exceeds 130 μg/l

   This can be done by adding the following line to the model:

   ```
   py.130 <- step(ypred - 130)    # indicator of whether ypred > 130
   ```

   Under the first model, the posterior prob that mean exceeds 130 is 0.37.

   Under both the second and third models, the posterior prob that mean exceeds 130 is about 0.5.

   (c) calculate the probability that a future THM concentration measured in the zone exceeds 130 μg/l

   This can be done by adding the following line to the model:

   ```
   pmean.130 <- step(theta - 130)    # indicator of whether zone mean > 130
   ```

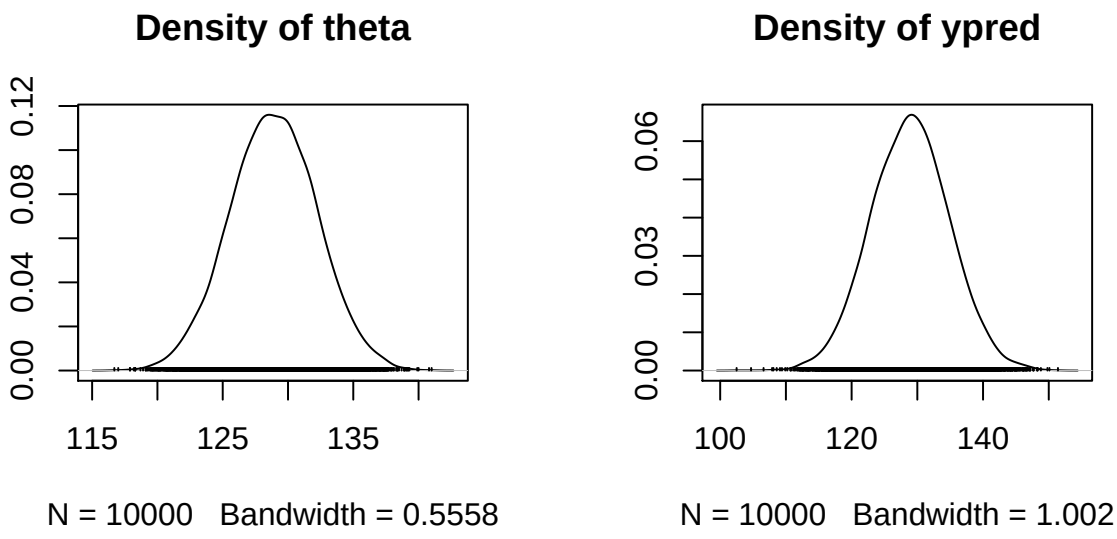
   Under the first model, the posterior prob that a future THM measurement exceeds 130 is 0.43.

   Under both the second and third model, the posterior prob that a future THM measurement exceeds 130 is about 0.5.

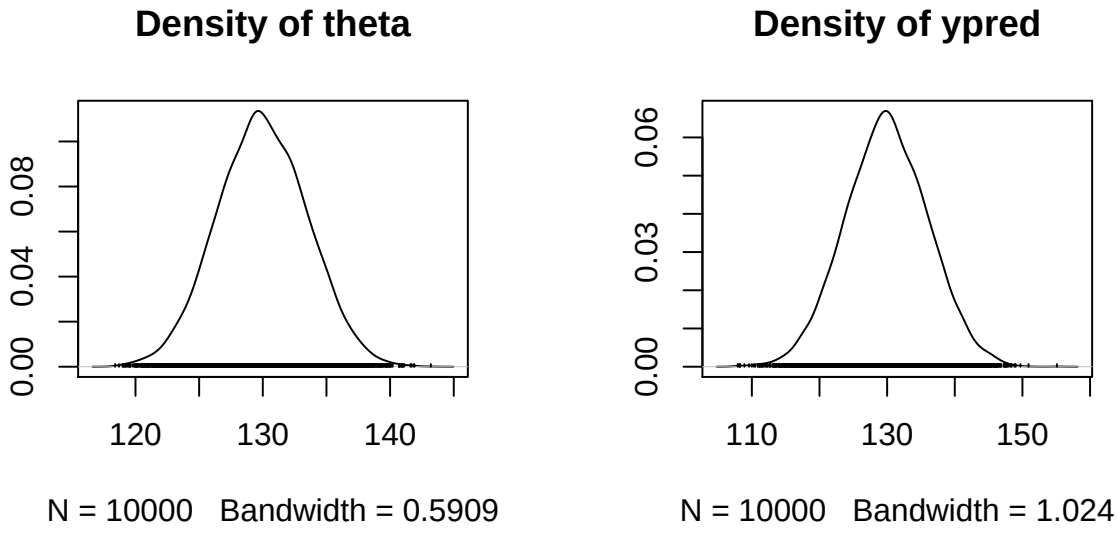
Note that the two uninformative priors give very similar results in this example.

Note that the posterior distribution of the mean, theta, and the predictive distribution for future THM levels (ypred) are centred about the same value, but the variance of the predictive distribution is much greater, reflecting sampling variation as well as posterior uncertainty about theta.

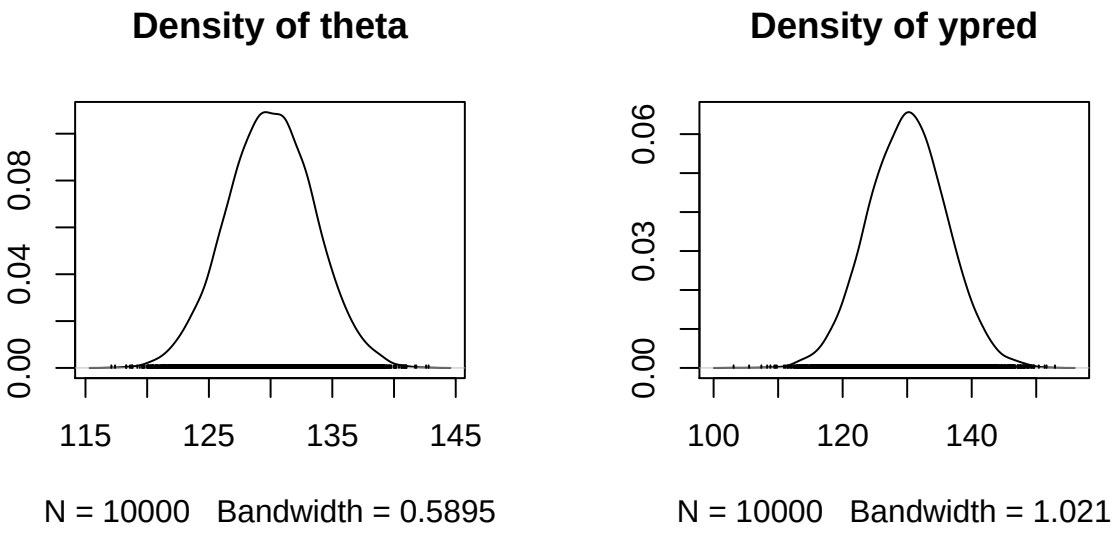
```
par(mfrow=c(1,2))
densplot(thm.B1.coda[,c("theta","ypred")])
```




```
densplot(thm.B2a.coda[,c("theta","ypred")])
```



```
densplot(thm.B2b.coda[,c("theta","ypred")])
```



Practical 3: Bayesian GLMs and non-linear regression

A. Dugongs

The data, one set of initial values and a basic model file are provided in the R lists `dugongs` and `dugongs_in1`, and in the file `dugongs-model.txt`.

1. Create a second set of initial values, and run the model to get the parameter estimates and model fit plot shown on slide 4-30.

   To use both sets of initial values, set the `inits` and `n.chains` arguments to `bugs()`

   `inits=list(dugongs_in1,dugongs_in2), n.chains=2`

   Remember to discard enough samples as burn-in before you start monitoring parameters. To run 40,000 iterations in total, discarding the first 10,000 burn-in, set `n.iter=40000, n.burnin=10000` as arguments to the `bugs()` function.

```
dugongs_in2_A1 <- list(alpha = 10, beta = 5, tau = 2, gamma = 0.2)
parameters.to.save <- c("alpha","beta","gamma","mu","sigma")

dugongs.A1.sim <- bugs(model="dugongs-model.txt",
                       data=dugongs,
                       inits=list(dugongs_in1, dugongs_in2_A1),
                       n.chains=2, # note 2 chains not 1 as before
                       n.burnin=10000,
                       n.iter=40000, # draw 40000, throw away first 10000
                       n.thin=1,
                       parameters.to.save=parameters.to.save,
                       DIC=FALSE)
dugongs.A1.coda <- as.mcmc.list(dugongs.A1.sim)
```

```
summary(dugongs.A1.coda[,c("alpha","beta","gamma","sigma")])
```

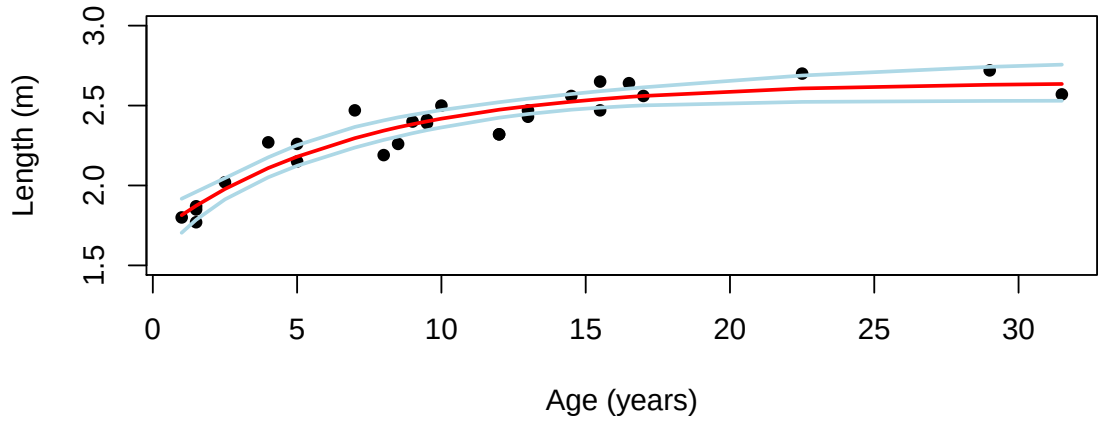
```
##
## Iterations = 10001:40000
## Thinning interval = 1
## Number of chains = 2
## Sample size per chain = 30000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##          Mean      SD  Naive SE Time-series SE
## alpha 2.65257 0.07231 2.952e-04      1.563e-03
## beta  0.97350 0.07644 3.121e-04      7.410e-04
```

```
## gamma 0.86228 0.03249 1.326e-04      6.047e-04
## sigma 0.09905 0.01517 6.195e-05      8.724e-05
##
## 2. Quantiles for each variable:
##
##          2.5%     25%    50%    75%  97.5%
## alpha 2.53100 2.60400 2.6455 2.6930 2.8150
## beta  0.82580 0.92390 0.9722 1.0220 1.1280
## gamma 0.78860 0.84440 0.8658 0.8840 0.9163
## sigma 0.07475 0.08826 0.0972 0.1078 0.1341
```

The R function `model.fit` (provided in `plot-functions.R`) draws model fit plots, e.g.

```
model.fit(dugongs$Y, dugongs$x, dugongs.A1.coda, "mu",
          pch=20, lwd=2, xlab="Age (years)", ylab="Length (m)",
          ylim=c(1.5,3))
```

```
model.fit(dugongs$Y, dugongs$x, dugongs.A1.coda, "mu",
          pch=20, lwd=2, xlab="Age (years)", ylab="Length (m)",
          ylim=c(1.5,3))
```



2. Change the prior to being uniform on $\log(\sigma)$: is there any influence?

   Note — you must always specify initial values for the stochastic parameters, i.e. the parameters assigned prior distributions, so you will need to edit the initial values files here as well as the model code. Also remember that you cannot specify the distribution of a function of a variable directly in the BUGS language. i.e. the following will *not* work:

   ```
   log(sigma) ~ dnorm(mu, tau)
   ```

   Instead you must create a new variable equal to $\log(\sigma)$ (called `log.sigma`, say), then assign

this new variable an appropriate prior, and then define `sigma` as a function of `log.sigma`. See solutions or ask if you are confused!

Note that you can use `log(.)`, `logit(.)`, `probit(.)` and `cloglog(.)` on the left hand side of a *deterministic* (`<-`) relation e.g. `log(sigma) <- log.sigma`, but only if `sigma` is not on the left hand side of any other expression.

```
dugongs_model_A2_file <- normalizePath("solutions/practical3/dugongs-model-A2.txt")
dugongs_in1_A2 <- list(alpha = 1, beta = 1, log.sigma = 1, gamma = 0.9)
dugongs_in2_A2 <- list(alpha = 1, beta = 1, log.sigma = 0.1, gamma = 0.9)
parameters.to.save <- c("alpha","beta","gamma","mu","sigma")

dugongs.A2.sim <- bugs(model=dugongs_model_A2_file,
                       data=dugongs,
                       inits=list(dugongs_in1_A2, dugongs_in2_A2),
                       n.chains=2,
                       n.burnin=10000,
                       n.iter=40000,
                       n.thin=1,
                       parameters.to.save=parameters.to.save,
                       DIC=FALSE)
dugongs.A2.coda <- as.mcmc.list(dugongs.A2.sim)
```

Results very similar to those with gamma prior on tau:

```
summary(dugongs.A2.coda[,c("alpha","beta","gamma","sigma")])
```

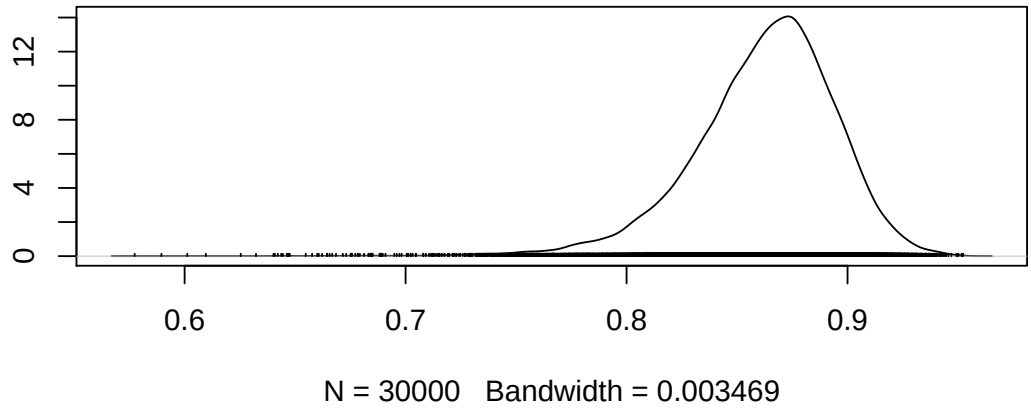
```
##
## Iterations = 10001:40000
## Thinning interval = 1
## Number of chains = 2
## Sample size per chain = 30000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##          Mean      SD  Naive SE Time-series SE
## alpha 2.65437 0.07425 3.031e-04      1.614e-03
## beta  0.97432 0.07759 3.168e-04      7.710e-04
## gamma 0.86301 0.03211 1.311e-04      5.887e-04
## sigma 0.09874 0.01498 6.114e-05      8.942e-05
##
## 2. Quantiles for each variable:
##
##          2.5%     25%     50%    75%  97.5%
## alpha 2.53200 2.60400 2.64700 2.6960 2.8190
```

```
## beta  0.82610 0.92320 0.97270 1.0230 1.1320
## gamma 0.79100 0.84470 0.86640 0.8848 0.9167
## sigma 0.07452 0.08814 0.09695 0.1074 0.1329
```

3. Check the posterior distribution for γ: do you think the uniform prior on this parameter is very influential?

   Using the model in question 1, we get the following posterior distribution for gamma:

```
densplot(dugongs.A1.coda[,"gamma"])
```

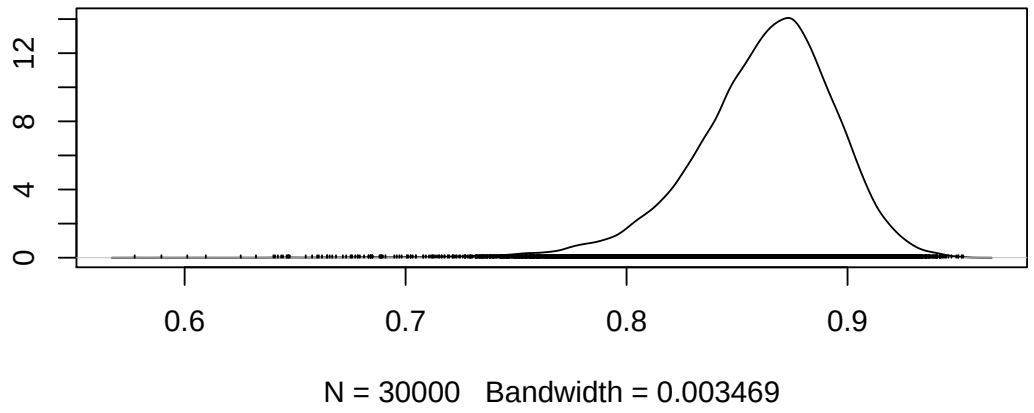


   Posterior mass is not too close to the limits (0, 1) imposed by the uniform prior on gamma, so doesn't appear to be too influential.

   Now try with a more informative prior: gamma $\sim$ dbeta(8,2)

```
dugongs_in2_A3 <- list(alpha = 10, beta = 5, tau = 2, gamma = 0.2)
parameters.to.save <- c("alpha","beta","gamma","mu","sigma")

dugongs.A3.sim <- bugs(model="dugongs-model.txt",
                    data=dugongs,
                    inits=list(dugongs_in1, dugongs_in2_A3),
                    n.chains=2, # note 2 chains not 1 as before
                    n.burnin=10000,
                    n.iter=40000,
                    n.thin=1,
                    parameters.to.save=parameters.to.save,
                    DIC=FALSE)
dugongs.A3.coda <- as.mcmc.list(dugongs.A3.sim)
```

```
densplot(dugongs.A3.coda[,"gamma"])
```



B. Beetles

The data, some initial values and a basic model file are provided in the R lists `beetles`, `beetles_in1` and `beetles_in2`; and in the file `beetles-model.txt`.

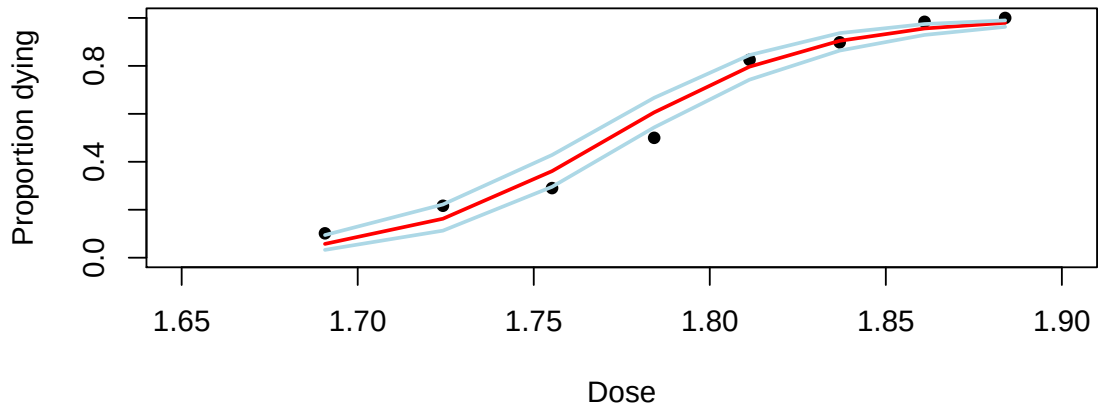
1. Run the model to get parameter estimates and a plot of the model fit (slide 4-24). Also look at history plots – have the chains converged?

```
parameters.to.save <- c("alpha","beta","p")

beetles.B1.sim <- bugs(model="beetles-model.txt",
                   data=beetles,
                   inits=list(beetles_in1,beetles_in2),
                   n.chains=2,
                   n.burnin=0, # start monitoring immediate to
                               # see good convergence of centred version
                   n.iter=40000,
                   n.thin=1,
                   parameters.to.save=parameters.to.save,
                   DIC=FALSE)
beetles.B1.coda <- as.mcmc.list(beetles.B1.sim)
```

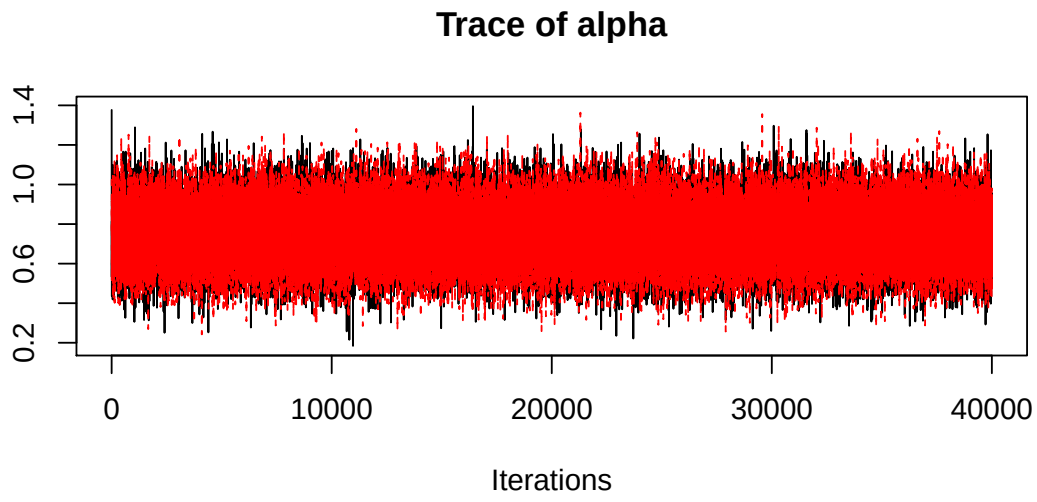
Note the good convergence of this centred version:

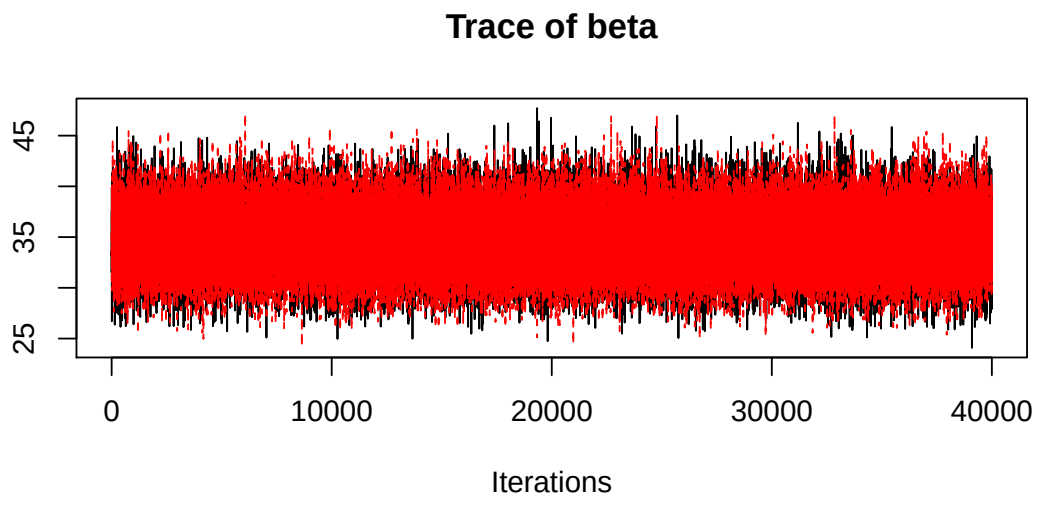
```
model.fit(x = beetles$x, y = beetles$r/beetles$n, beetles.B1.coda, "p",
          pch=20, lwd=2, xlab="Dose", ylab="Proportion dying",
```

```
          xlim = c(1.65, 1.9), ylim = c(0, 1))
```

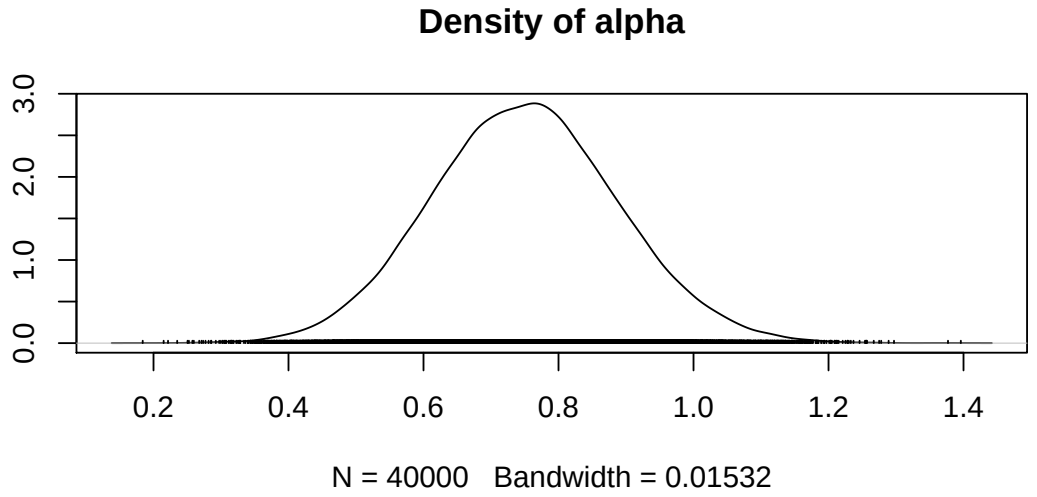


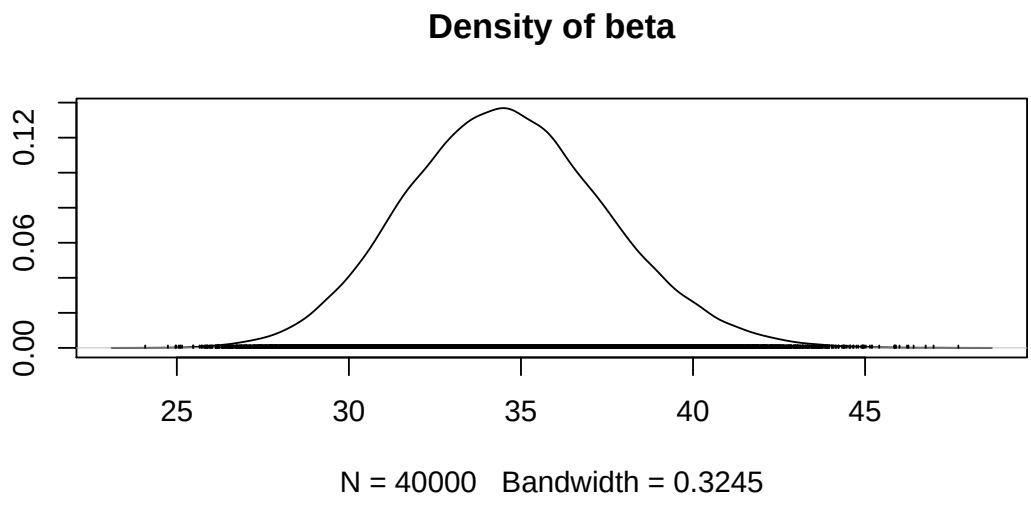

```
traceplot(beetles.B1.coda[,c("alpha","beta")])
```






```
densplot(beetles.B1.coda[,c("alpha","beta")])
```





2. Try 'uncentering' the covariate and check the influence on the convergence of `beta`. You might need to make the initial conditions less extreme to avoid numerical issues.

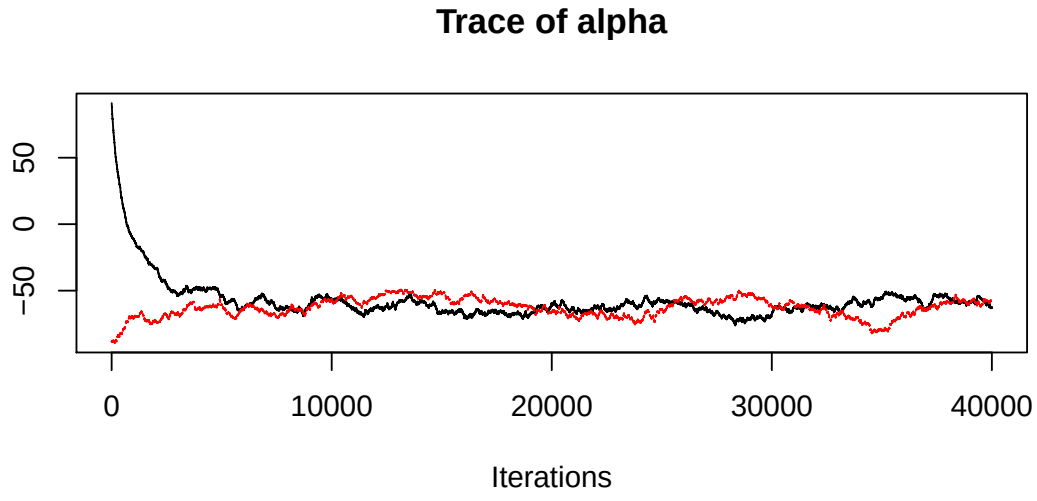
   Use the following specification for `logit(p)`:

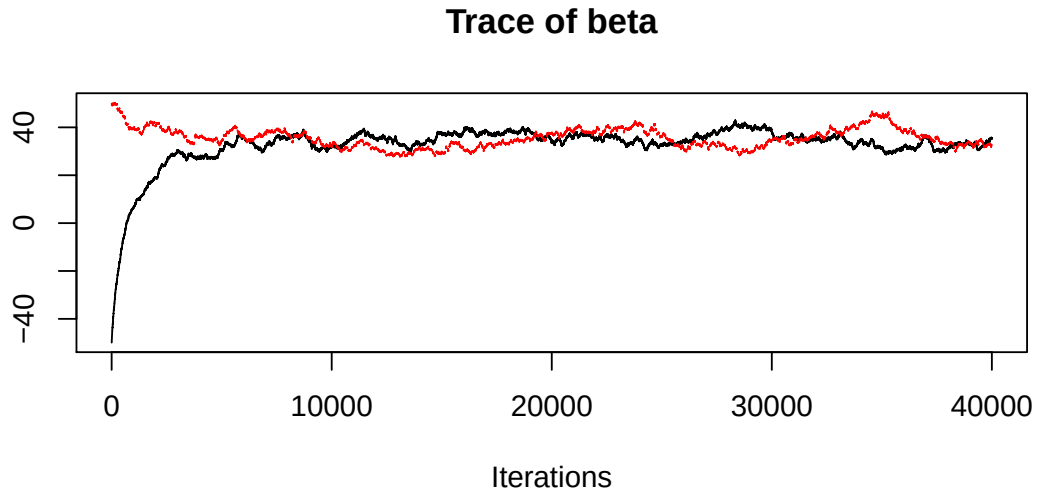
```
# uncentered version - poor convergence
logit(p[i]) <- alpha + beta * x[i]
```

```
beetles_model_B2_file <- normalizePath("solutions/practical3/beetles-model-B2.txt")
parameters.to.save <- c("alpha","beta","p")

beetles.B2.sim <- bugs(model=beetles_model_B2_file,
                   data=beetles,
                   inits=list(beetles_in1,beetles_in2),
                   n.chains=2,
                   n.burnin=0, # start monitoring immediate to
                               # see poor convergence of uncentred version
                   n.iter=40000,
                   n.thin=1,
                   parameters.to.save=parameters.to.save,
                   DIC=FALSE)
beetles.B2.coda <- as.mcmc.list(beetles.B2.sim)
```

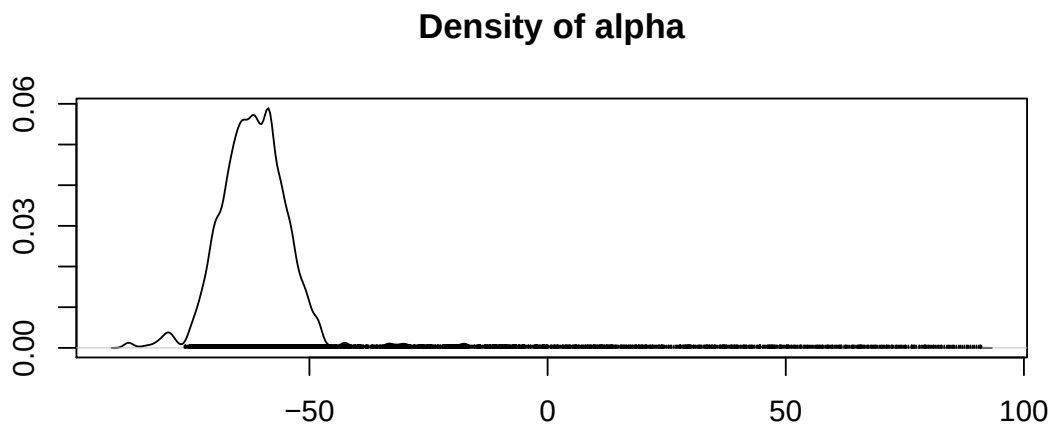
   Notice the poor convergence of this uncentred version:

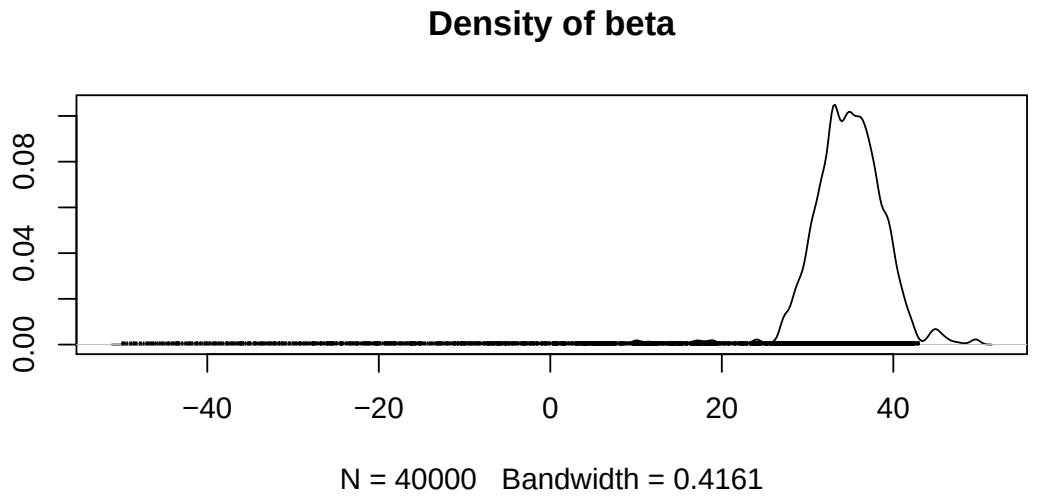
```
traceplot(beetles.B2.coda[,c("alpha","beta")])
```






```
densplot(beetles.B2.coda[,c("alpha","beta")])
```





3. Try different link functions: complementary log-log [$\log(-\log(1-p))$] and probit [$\Phi(p)$]. These can be very sensitive to initial values! Try the two different ways of implementing the probit model as described in the lecture notes (slide 4-27).

```
beetles_model_B3a_file <- normalizePath("solutions/practical3/beetles-model-B3a.txt
beetles_in1_B3a <- list(alpha=0, beta=0)
beetles_in2_B3a <- list(alpha=1, beta=1)
parameters.to.save <- c("alpha","beta","p")

beetles.B3a.sim <- bugs(model=beetles_model_B3a_file,
                        data=beetles,
                        inits=list(beetles_in1_B3a, beetles_in2_B3a),
```

```
                        n.chains=2,
                        n.burnin=1000,
                        n.iter=11000,
                        n.thin=1,
                        parameters.to.save=parameters.to.save,
                        DIC=FALSE)
beetles.B3a.coda <- as.mcmc.list(beetles.B3a.sim)
```

With a cloglog link:

```
summary(beetles.B3a.coda[,c("alpha","beta")])
```

```
##
## Iterations = 1001:11000
## Thinning interval = 1
## Number of chains = 2
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean      SD  Naive SE Time-series SE
## alpha -0.04517 0.07987 0.0005648      0.0005705
## beta  22.20580 1.79401 0.0126856      0.0130488
##
## 2. Quantiles for each variable:
##
##          2.5%      25%      50%     75%   97.5%
## alpha -0.2032 -0.09882 -0.04497  0.0084  0.1105
## beta  18.8500 20.95750 22.16000 23.4200 25.7900
```

```
beetles_model_B3b_file <- normalizePath("solutions/practical3/beetles-model-B3b.txt
beetles_in1_B3b <- list(alpha=0, beta=30)
beetles_in2_B3b <- list(alpha=1, beta=0)
parameters.to.save <- c("alpha","beta","p")

beetles.B3b.sim <- bugs(model=beetles_model_B3b_file,
                        data=beetles,
                        inits=list(beetles_in1_B3b, beetles_in2_B3b),
                        n.chains=2,
                        n.burnin=1000,
                        n.iter=11000,
                        n.thin=1,
                        parameters.to.save=parameters.to.save,
                        DIC=FALSE)
```

```
beetles.B3b.coda <- as.mcmc.list(beetles.B3b.sim)
```

With a probit (standard implementation) link:

```
summary(beetles.B3b.coda[,c("alpha","beta")])
```

```
##
## Iterations = 1001:11000
## Thinning interval = 1
## Number of chains = 2
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##          Mean      SD  Naive SE Time-series SE
## alpha  0.4474 0.07794 0.0005511      0.0006086
## beta  19.8364 1.48561 0.0105049      0.0112066
##
## 2. Quantiles for each variable:
##
##          2.5%     25%     50%     75%   97.5%
## alpha  0.2983  0.3938  0.4467  0.4994  0.6005
## beta  17.0100 18.8300 19.8000 20.8200 22.8500
```

```
beetles_model_B3c_file <- normalizePath("solutions/practical3/beetles-model-B3c.txt
beetles_in1_B3b <- list(alpha=0, beta=30)
beetles_in2_B3b <- list(alpha=1, beta=0)
parameters.to.save <- c("alpha","beta","p")

beetles.B3c.sim <- bugs(model=beetles_model_B3c_file,
                        data=beetles,
                        inits=list(beetles_in1_B3b, beetles_in2_B3b),
                        n.chains=2,
                        n.burnin=1000,
                        n.iter=11000,
                        n.thin=1,
                        parameters.to.save=parameters.to.save,
                        DIC=FALSE)
beetles.B3c.coda <- as.mcmc.list(beetles.B3c.sim)
```

With a probit (alternative implementation) link:

```
summary(beetles.B3c.coda[,c("alpha","beta")])
```

```
##
```

```
## Iterations = 1001:11000
## Thinning interval = 1
## Number of chains = 2
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##          Mean      SD  Naive SE Time-series SE
## alpha  0.4474 0.07794 0.0005511      0.0006086
## beta  19.8364 1.48561 0.0105049      0.0112066
##
## 2. Quantiles for each variable:
##
##          2.5%     25%     50%     75%   97.5%
## alpha  0.2983  0.3938  0.4467  0.4994  0.6005
## beta  17.0100 18.8300 19.8000 20.8200 22.8500
```

Practical 4: Predictions, model checking and comparison

A. Dugongs

1. Run the dugongs model from practical 3 and include statements to calculate:
   - standardised residuals
   - p-value for each residual
   - predictive distribution for each dugong, `Y.pred[i]`
   - Bayesian p-value calculating posterior probability that `Y[i]` > `Y.pred[i]`

   You can either edit the model code in `dugongs-model.txt` to add in the relevant statements to calculate residuals and p-values (and the corresponding script to set appropriate monitors), or use the pre-prepared code in `dugongs-check-model.txt` .

   We can calculate the above quantities using:

```
# standardised residuals
res[i] <- (Y[i] - mu[i]) / sigma

# p-value for res
P.res[i] <- phi(res[i])

# predicted response for dugong i
Y.pred[i] ~ dnorm(mu[i], tau)

# indicator of whether observed > predicted
# (posterior mean on P.pred = bayesian p-value)
P.pred[i]  <-  step(Y[i] - Y.pred[i])
```

```
# create a second set of initial values
dugongs_in2_A1 <- list(alpha = 10, beta = 5, tau = 2, gamma = 0.2)

parameters.to.save <- c("alpha","beta","gamma","mu","sigma","res","P.res",
                        "P.pred","Y.pred")
dugongs.A1.sim <- bugs(model="dugongs-check-model.txt",
                       data=dugongs,
                       inits=list(dugongs_in1, dugongs_in2_A1),
                       n.chains=2,
                       n.burnin=5000,
                       n.iter=55000,
                       n.thin=1,
                       parameters.to.save=parameters.to.save,
                       DIC=FALSE)
```

```
dugongs.A1.coda <- as.mcmc.list(dugongs.A1.sim)
```

```
summary(dugongs.A1.coda[,c("alpha","beta","gamma","sigma")])
```

```
##
## Iterations = 5001:55000
## Thinning interval = 1
## Number of chains = 2
## Sample size per chain = 50000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##          Mean      SD  Naive SE Time-series SE
## alpha 2.65295 0.07318 2.314e-04      1.213e-03
## beta  0.97392 0.07683 2.429e-04      6.010e-04
## gamma 0.86235 0.03292 1.041e-04      4.856e-04
## sigma 0.09907 0.01504 4.757e-05      6.752e-05
##
## 2. Quantiles for each variable:
##
##          2.5%     25%     50%    75%  97.5%
## alpha 2.52900 2.60400 2.64600 2.6940 2.8200
## beta  0.82670 0.92320 0.97220 1.0230 1.1300
## gamma 0.78720 0.84450 0.86570 0.8847 0.9167
## sigma 0.07478 0.08845 0.09733 0.1078 0.1332
```

2. Try reproducing the box plots of standardised residuals and the 'QQ plot' of residual p-values against order-statistics shown in the lecture notes (similar those in slide 5-10), plus the plot showing the predictive fit of the model (slide 5-6). Also produce a 'QQ plot' of the predictive p-values. Compare the p-values calculated using the two methods.

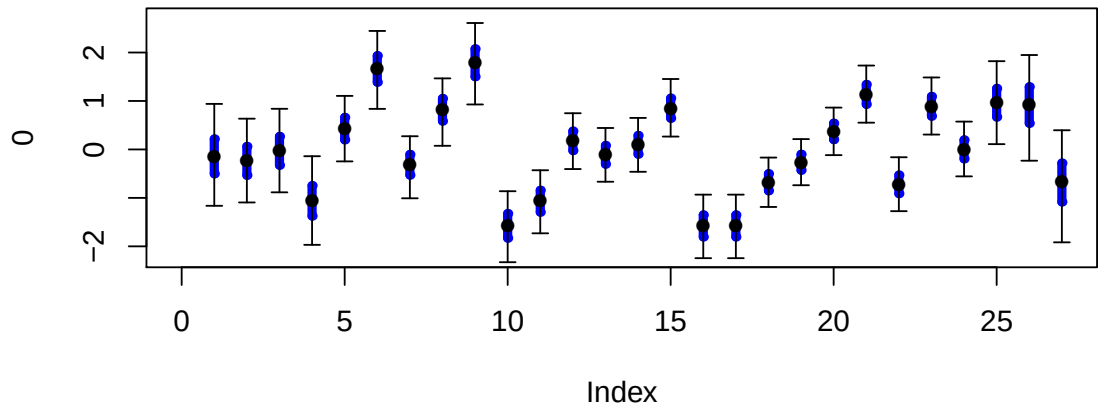
   Produce plots of residuals versus fitted values and of residuals versus covariates as well.

   You may find the bundled plotting functions (in `plot-functions.R`) helpful:

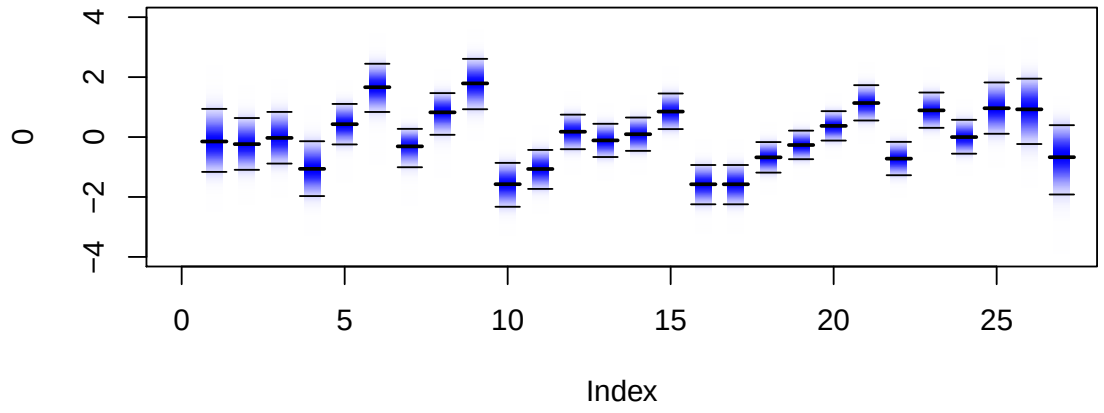
   ```
   bugs.boxplot(dugongs.A1.coda, "res")
   density.strips(dugongs.A1.coda, "res")
   catplot(dugongs.A1.coda, "P.res", ordered=TRUE)
   ```

   Box plot of standardised residuals:

```
bugs.boxplot(dugongs.A1.coda, "res")
```

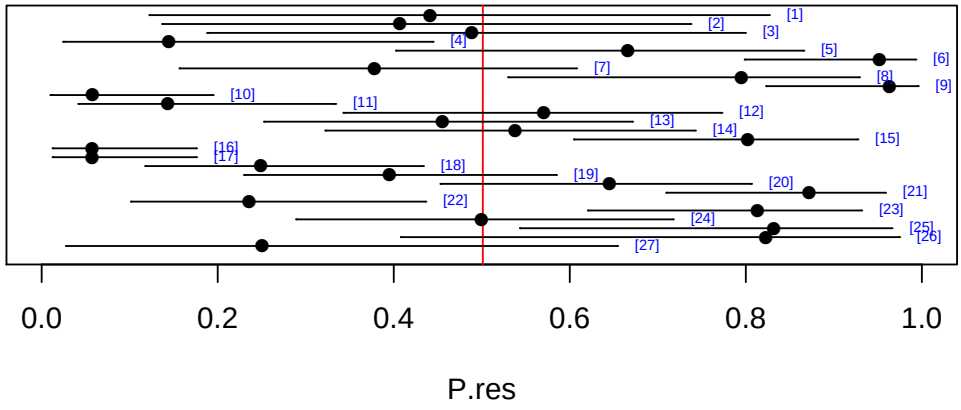



```
density.strips(dugongs.A1.coda, "res")
```

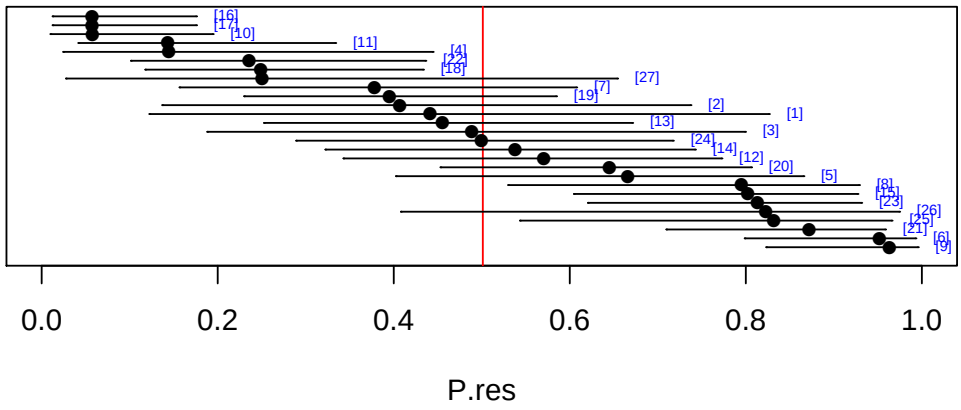


P-values against order statistics:

```
catplot(dugongs.A1.coda, "P.res")
```

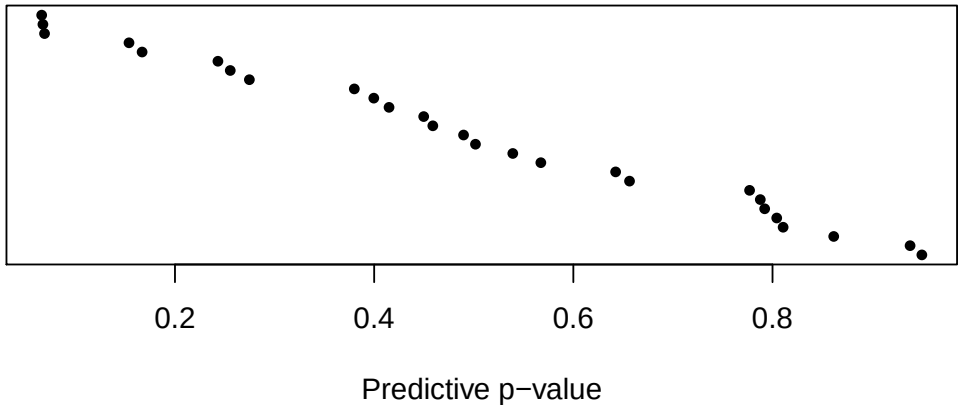



```
catplot(dugongs.A1.coda, "P.res", ordered=TRUE)
```



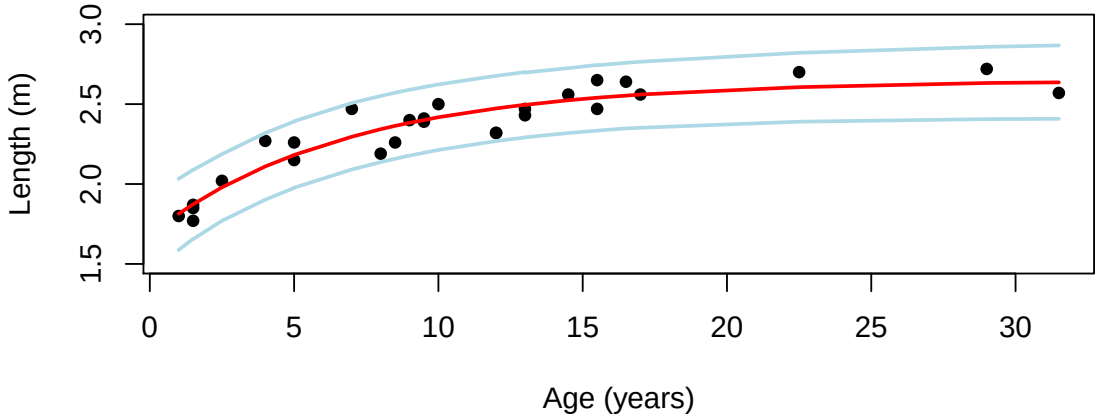
Predictive p-values against order statistics:

```
res <- get.coda.matrix(dugongs.A1.coda, "P.pred")
qs <- apply(res, 2, mean)
plot(sort(qs), length(qs):1, pch=20, yaxt="n",
     xlab="Predictive p-value", ylab="")
```



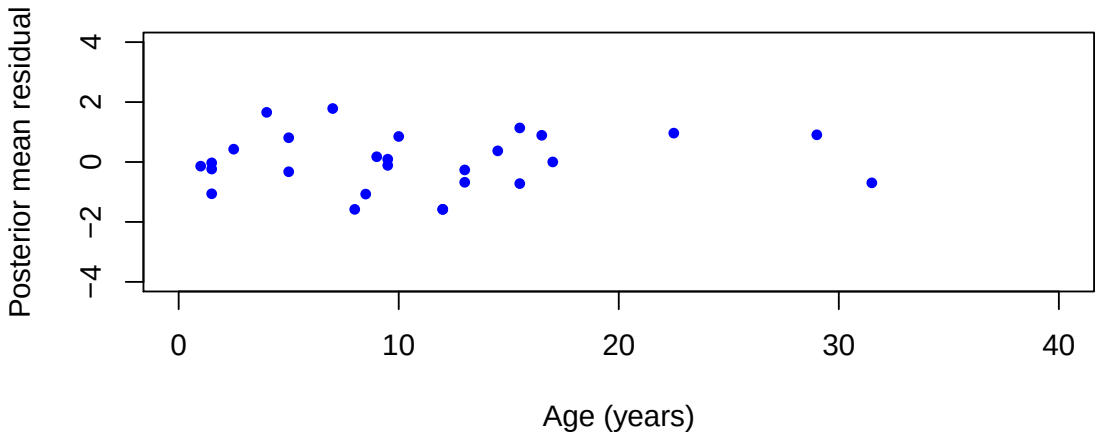
Predictive model fit:

```
model.fit(dugongs$Y, dugongs$x, dugongs.A1.coda, "Y.pred",
          pch=20, lwd=2, xlab="Age (years)", ylab="Length (m)",
          ylim=c(1.5,3))
```



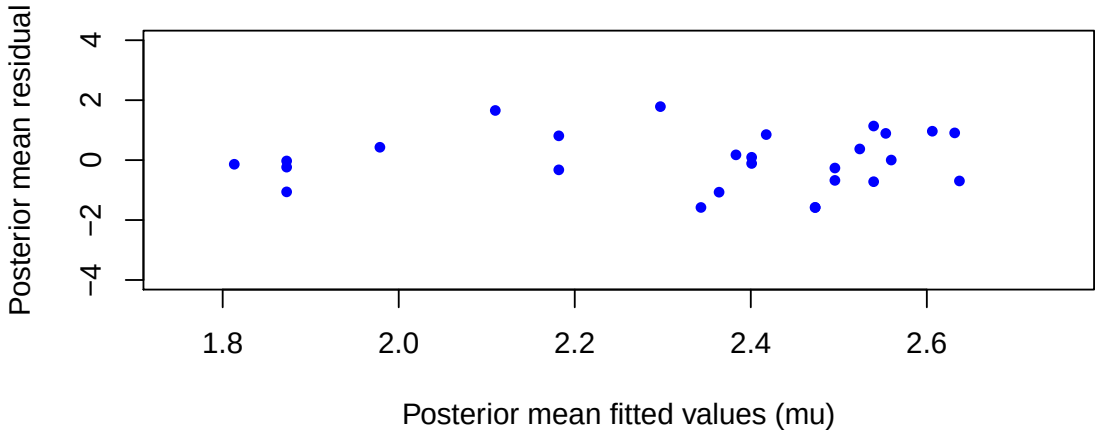
Residuals against covariate:

```
res <- get.coda.matrix(dugongs.A1.coda, "res")
plot(dugongs$x, colMeans(res), pch=20, ylim=c(-4,4), xlim=c(0,40),
     col="blue", xlab="Age (years)", ylab="Posterior mean residual")
```



Residuals against fitted values:

```
res <- get.coda.matrix(dugongs.A1.coda, "res")
fit <- get.coda.matrix(dugongs.A1.coda, "mu")
plot(colMeans(fit), colMeans(res), pch=20, col="blue",
     ylim=c(-4, 4), xlim=c(1.75, 2.75),
     xlab="Posterior mean fitted values (mu)", ylab="Posterior mean residual")
```



A slight suggestion that variance increases with Y

3. Edit the code to include statements to predict dugong length at ages 35 and 40. Try producing the plot of these predictions similar to that shown on slide 5-5.

   We can make the predictions by adding 35 and 40 to the end of the `x` vector; adding `NA`,

NA to the end of the Y vector; and increasing N to 29.

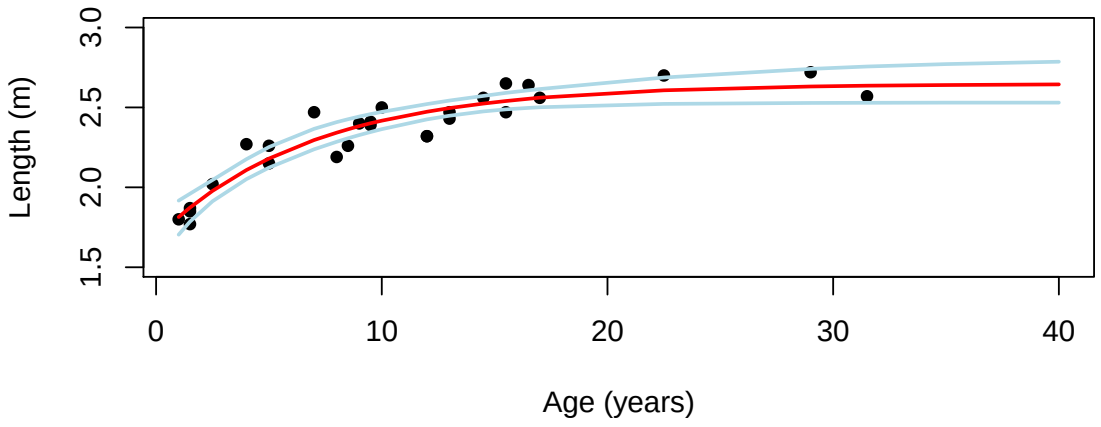
```
# create a second set of initial values
dugongs_in2_A1 <- list(alpha = 10, beta = 5, tau = 2, gamma = 0.2)

# edit the data to make predictions at age 35 and 40
dugongs_A3 <- dugongs
dugongs_A3$x <- c(dugongs_A3$x, 35, 40)
dugongs_A3$Y <- c(dugongs_A3$Y, NA, NA)
dugongs_A3$N <- 29

parameters.to.save <- c("alpha","beta","gamma","mu","sigma","res","P.res",
                        "P.pred","Y.pred", "Y[28]","Y[29]")
dugongs.A3.sim <- bugs(model="dugongs-check-model.txt",
                       data=dugongs_A3,
                       inits=list(dugongs_in1, dugongs_in2_A1),
                       n.chains=2,
                       n.burnin=5000,
                       n.iter=55000,
                       n.thin=1,
                       parameters.to.save=parameters.to.save,
                       DIC=FALSE)
dugongs.A3.coda <- as.mcmc.list(dugongs.A3.sim)
```

```
summary(dugongs.A3.coda[,c("Y[28]","Y[29]")])
```

```
##
## Iterations = 5001:55000
## Thinning interval = 1
## Number of chains = 2
## Sample size per chain = 50000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##        Mean     SD  Naive SE Time-series SE
## Y[28] 2.643 0.1175 0.0003715        0.00100
## Y[29] 2.648 0.1192 0.0003769        0.00106
##
## 2. Quantiles for each variable:
##
##        2.5%   25%   50%   75% 97.5%
## Y[28] 2.412 2.566 2.642 2.720 2.876
## Y[29] 2.414 2.569 2.646 2.725 2.886
```

Model fit - note the additional points at $x = 35$ and $x = 40$:

```
model.fit(dugongs_A3$Y, dugongs_A3$x, dugongs.A3.coda, "mu",
          pch=20, lwd=2, xlab="Age (years)", ylab="Length (m)",
          ylim=c(1.5,3))
```



4. The R list `dugongs_outl` contains a modified version of the data with one observation changed to be an outlier. Repeat the above analyses using this new data file (edit the scripts accordingly).

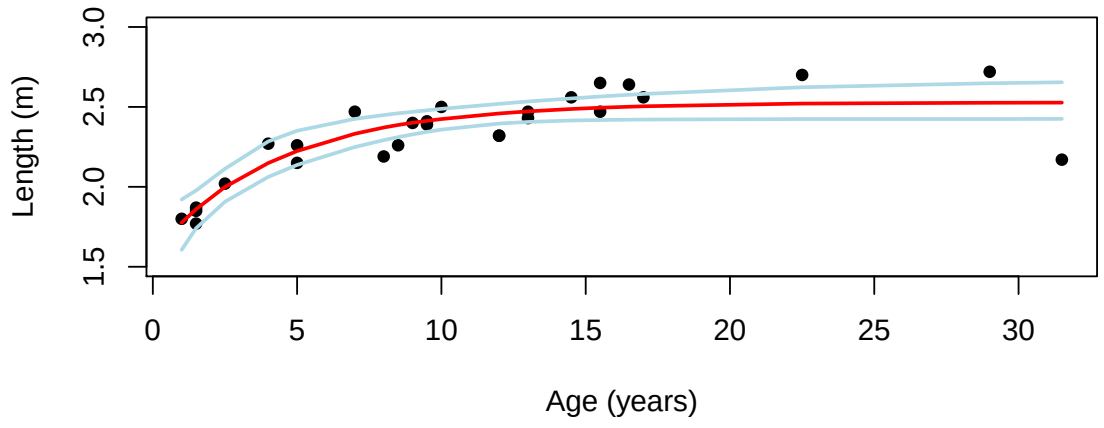
```
# create a second set of initial values
dugongs_in2_A4 <- list(alpha = 10, beta = 5, tau = 2, gamma = 0.2)

parameters.to.save <- c("alpha","beta","gamma","mu","sigma","res","P.res",
                        "P.pred","Y.pred")
dugongs.A4.sim <- bugs(model="dugongs-check-model.txt",
                       data=dugongs_outl,
                       inits=list(dugongs_in1, dugongs_in2_A4),
                       n.chains=2,
                       n.burnin=5000,
                       n.iter=55000,
                       n.thin=1,
                       parameters.to.save=parameters.to.save,
                       DIC=FALSE)
dugongs.A4.coda <- as.mcmc.list(dugongs.A4.sim)
```

```
summary(dugongs.A4.coda[,c("alpha","beta","gamma","sigma")])

##
## Iterations = 5001:55000
```

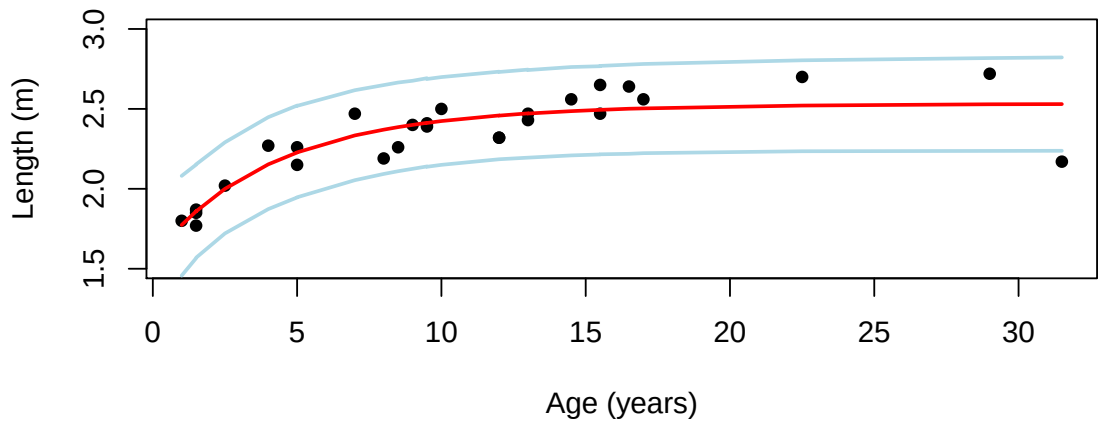
```
## Thinning interval = 1
## Number of chains = 2
## Sample size per chain = 50000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##         Mean      SD  Naive SE Time-series SE
## alpha 2.5329 0.06262 1.980e-04      7.029e-04
## beta  0.9763 0.15629 4.942e-04      1.423e-03
## gamma 0.7860 0.07060 2.233e-04      8.439e-04
## sigma 0.1334 0.02041 6.455e-05      9.176e-05
##
## 2. Quantiles for each variable:
##
##         2.5%    25%    50%    75%  97.5%
## alpha 2.4250 2.4900 2.5280 2.5700 2.6710
## beta  0.7391 0.8814 0.9576 1.0450 1.3330
## gamma 0.6079 0.7536 0.7986 0.8337 0.8861
## sigma 0.1007 0.1189 0.1310 0.1451 0.1805
```

Note that the residual SD has increased as a result of the outlier

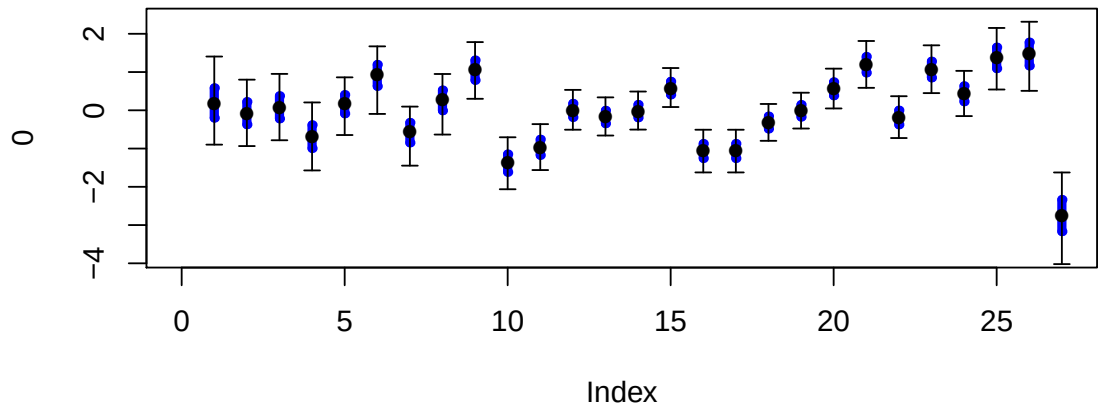
```
model.fit(dugongs_outl$Y, dugongs_outl$x, dugongs.A4.coda, "mu",
          pch=20, lwd=2, xlab="Age (years)", ylab="Length (m)",
          ylim=c(1.5,3))
```



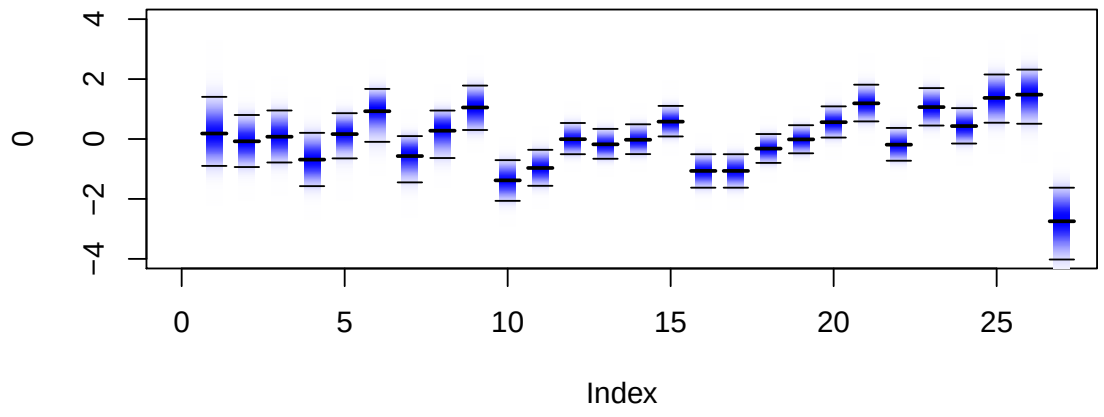
```
model.fit(dugongs_outl$Y, dugongs_outl$x, dugongs.A4.coda, "Y.pred",
          pch=20, lwd=2, xlab="Age (years)", ylab="Length (m)",
          ylim=c(1.5,3))
```



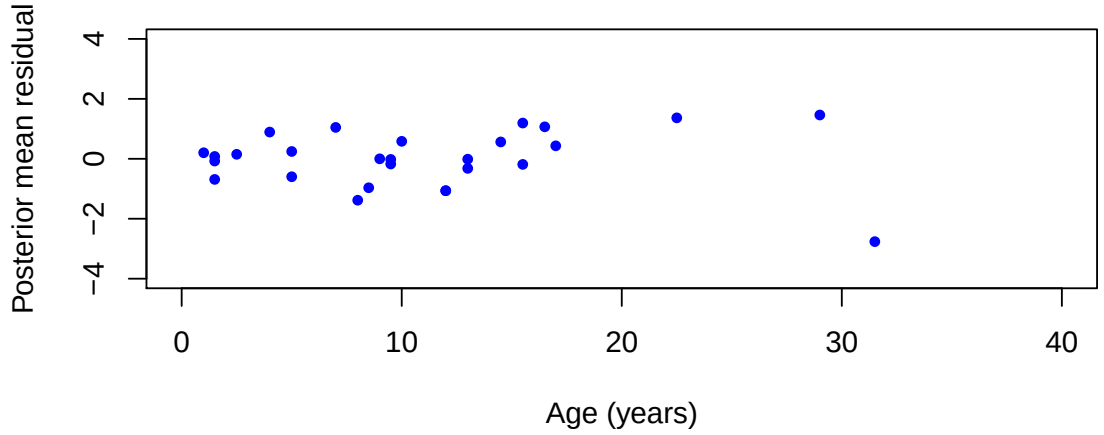

```
bugs.boxplot(dugongs.A4.coda, "res")
```




```
density.strips(dugongs.A4.coda, "res")
```

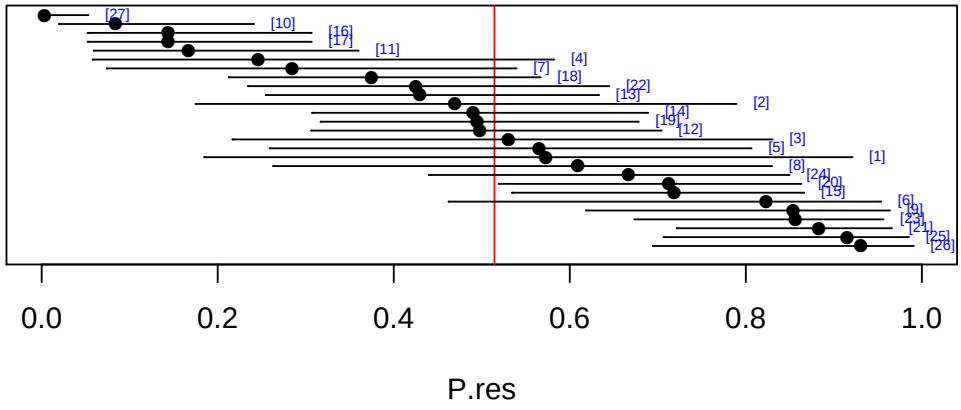



```
res <- get.coda.matrix(dugongs.A4.coda, "res")
plot(dugongs_outl$x, colMeans(res), pch=20, col="blue",
     ylim=c(-4,4), xlim=c(0,40),
     xlab="Age (years)", ylab="Posterior mean residual")
```



Note the large negative residual corresponding to the outlier.

```
catplot(dugongs.A4.coda, "P.res", ordered=TRUE)
```




```
P.res.names <- paste0("P.res[", 1:27,"]")
summary(dugongs.A4.coda[, P.res.names])

##
## Iterations = 5001:55000
## Thinning interval = 1
## Number of chains = 2
## Sample size per chain = 50000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##               Mean      SD  Naive SE Time-series SE
## P.res[1]  0.567655 0.19872 6.284e-04      0.0014521
## P.res[2]  0.472948 0.16145 5.105e-04      0.0008144
## P.res[3]  0.528820 0.16139 5.103e-04      0.0008151
## P.res[4]  0.265665 0.13788 4.360e-04      0.0006898
## P.res[5]  0.556769 0.14039 4.440e-04      0.0008500
## P.res[6]  0.793689 0.12556 3.970e-04      0.0011755
## P.res[7]  0.289475 0.12218 3.864e-04      0.0010769
## P.res[8]  0.591075 0.14704 4.650e-04      0.0013826
## P.res[9]  0.836214 0.09067 2.867e-04      0.0007853
## P.res[10] 0.095831 0.05793 1.832e-04      0.0003782
## P.res[11] 0.178002 0.07802 2.467e-04      0.0005046
## P.res[12] 0.499854 0.10278 3.250e-04      0.0006692
## P.res[13] 0.433282 0.09724 3.075e-04      0.0005590
## P.res[14] 0.492216 0.09820 3.105e-04      0.0005688
## P.res[15] 0.713751 0.08535 2.699e-04      0.0004371
```

```
## P.res[16] 0.153002 0.06582 2.081e-04      0.0003285
## P.res[17] 0.153002 0.06582 2.081e-04      0.0003285
## P.res[18] 0.378330 0.09103 2.878e-04      0.0005097
## P.res[19] 0.495132 0.09258 2.927e-04      0.0004976
## P.res[20] 0.706792 0.08834 2.794e-04      0.0005845
## P.res[21] 0.872621 0.06356 2.010e-04      0.0004213
## P.res[22] 0.428731 0.10583 3.347e-04      0.0008523
## P.res[23] 0.845449 0.07349 2.324e-04      0.0005540
## P.res[24] 0.660791 0.10608 3.354e-04      0.0009234
## P.res[25] 0.896800 0.07316 2.313e-04      0.0006936
## P.res[26] 0.907578 0.07831 2.476e-04      0.0008082
## P.res[27] 0.008735 0.01529 4.834e-05      0.0001259
##
## 2. Quantiles for each variable:
##
##                2.5%       25%      50%      75%   97.5%
## P.res[1]  1.847e-01 0.4204000 0.572600 0.720400 0.92010
## P.res[2]  1.749e-01 0.3543000 0.469500 0.588400 0.78840
## P.res[3]  2.168e-01 0.4129000 0.530400 0.646900 0.82940
## P.res[4]  5.796e-02 0.1599000 0.246000 0.350900 0.58140
## P.res[5]  2.590e-01 0.4650000 0.565200 0.657700 0.80550
## P.res[6]  4.623e-01 0.7353750 0.823100 0.884100 0.95270
## P.res[7]  7.399e-02 0.1980000 0.284700 0.374100 0.53850
## P.res[8]  2.629e-01 0.4968000 0.609000 0.700800 0.82880
## P.res[9]  6.183e-01 0.7841000 0.853600 0.905000 0.96270
## P.res[10] 1.950e-02 0.0529300 0.083780 0.125800 0.24030
## P.res[11] 5.911e-02 0.1201000 0.166800 0.224000 0.35900
## P.res[12] 3.060e-01 0.4272000 0.497800 0.570800 0.70330
## P.res[13] 2.548e-01 0.3644000 0.429600 0.498500 0.63230
## P.res[14] 3.072e-01 0.4233000 0.490200 0.559400 0.68810
## P.res[15] 5.342e-01 0.6577000 0.718700 0.775000 0.86540
## P.res[16] 5.218e-02 0.1046000 0.143700 0.191700 0.30580
## P.res[17] 5.218e-02 0.1046000 0.143700 0.191700 0.30580
## P.res[18] 2.125e-01 0.3137750 0.374500 0.438700 0.56580
## P.res[19] 3.169e-01 0.4310000 0.494800 0.558400 0.67730
## P.res[20] 5.193e-01 0.6493000 0.712600 0.770500 0.86190
## P.res[21] 7.215e-01 0.8366000 0.883000 0.919600 0.96500
## P.res[22] 2.342e-01 0.3532000 0.425000 0.500300 0.64380
## P.res[23] 6.735e-01 0.8025000 0.856500 0.899900 0.95530
## P.res[24] 4.399e-01 0.5895000 0.666800 0.737800 0.84870
## P.res[25] 7.066e-01 0.8625000 0.915000 0.950400 0.98430
## P.res[26] 6.945e-01 0.8775000 0.930500 0.962400 0.98970
## P.res[27] 2.903e-05 0.0007646 0.003026 0.009773 0.05213
```

5. How could you modify the model specification to accommodate the outlier in the above analysis? How do the diagnostics change when you do this? Note: the standard deviation of a t distribution with precision parameter τ and ν degrees of freedom is $\sqrt{\frac{\nu}{(\nu-2)\tau}}$.

   Replace Normal likelihood with robust t distribution on 4 df

```
model {
for (i in 1:N){
  Y[i] ~ dt(mu[i], tau, 4)
  mu[i] <- alpha - beta * pow(gamma, x[i])

  # standardised residuals
  res[i] <- (Y[i] - mu[i]) / sigma

  # p-value for res (note: not correct under this
  # model, since residuals non-normal)
  P.res[i] <- phi(res[i])

  Y.pred[i] ~ dnorm(mu[i], tau)
  P.pred[i]  <-  step(Y[i] - Y.pred[i])
}
alpha ~ dunif(0, 100)
beta ~ dunif(0, 100)
gamma ~ dunif(0, 1.0)

tau ~ dgamma(0.001, 0.001)

# standard deviation of t-4 errors
sigma <- sqrt(2/tau)
}
```

```
dugongs_model_A5_file <- normalizePath("solutions/practical4/dugongs-model-A5.txt")
dugongs_in2_A5 <- list(alpha = 10, beta = 5, tau = 2, gamma = 0.2)
parameters.to.save <- c("alpha","beta","gamma","mu","sigma","res","P.res",
                        "P.pred","Y.pred")

dugongs.A5.sim <- bugs(model=dugongs_model_A5_file,
                       data=dugongs_outl,
                       inits=list(dugongs_in1, dugongs_in2_A5),
                       n.chains=2,
                       n.burnin=5000,
                       n.iter=55000,
                       n.thin=1,
                       parameters.to.save=parameters.to.save,
                       DIC=FALSE)
```

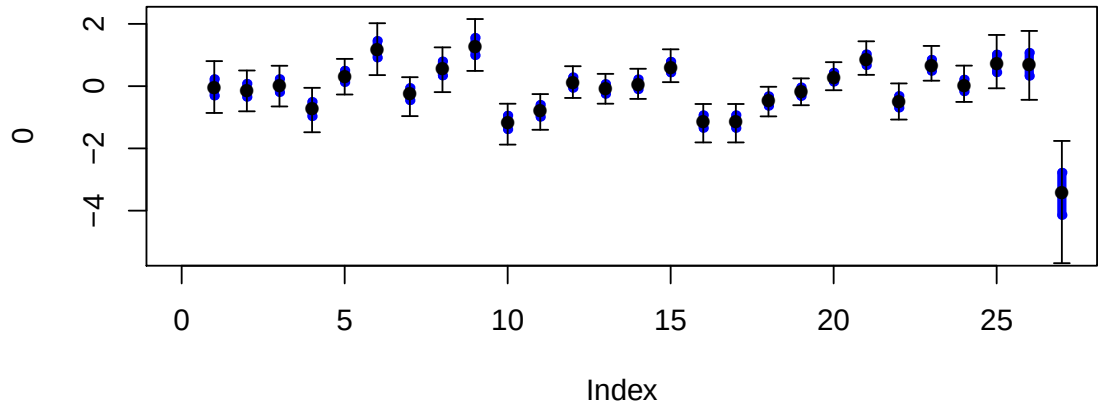
```
dugongs.A5.coda <- as.mcmc.list(dugongs.A5.sim)
```

```
summary(dugongs.A5.coda[,c("P.res[27]")])
```

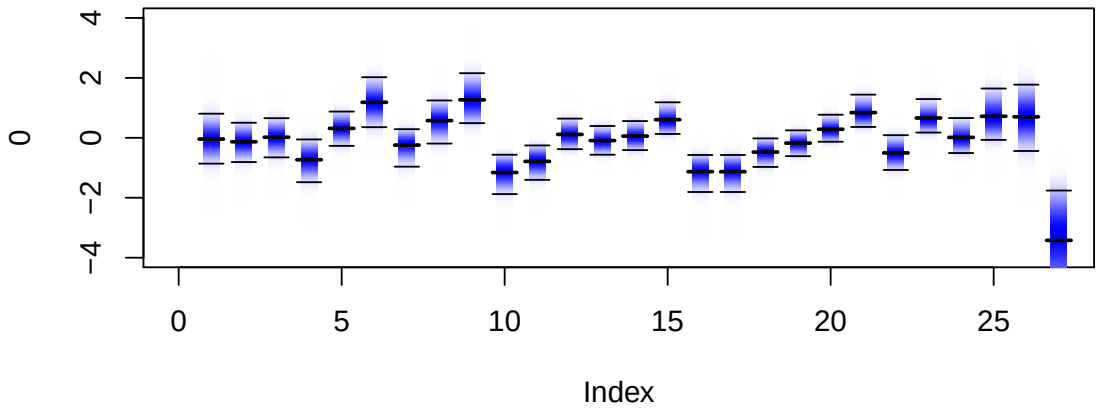
```
##
## Iterations = 5001:55000
## Thinning interval = 1
## Number of chains = 2
## Sample size per chain = 50000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean             SD       Naive SE Time-series SE
##      4.526e-03      1.252e-02      3.959e-05      1.556e-04
##
## 2. Quantiles for each variable:
##
##      2.5%       25%       50%       75%     97.5%
## 6.416e-09 1.700e-05 3.108e-04 2.778e-03 3.916e-02
```

The final point still appears to be an outlier, and still has a small predictive p-value than under the normal errors model (Note that calculation of P.res is no longer valid for this model because the residuals are not Normal)

```
bugs.boxplot(dugongs.A5.coda, "res")
```



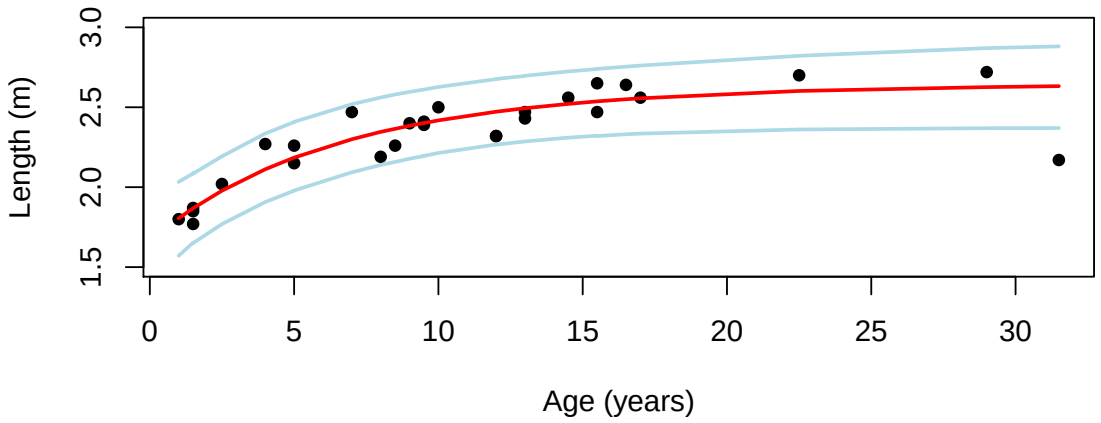
```
density.strips(dugongs.A5.coda, "res")
```



However, notice that the 95% predictive interval is actually slightly narrower under the robust model than the Normal errors model. The final point still falls outside the prediction interval, but this outlier is now having less influence on the fit to the remaining data points:

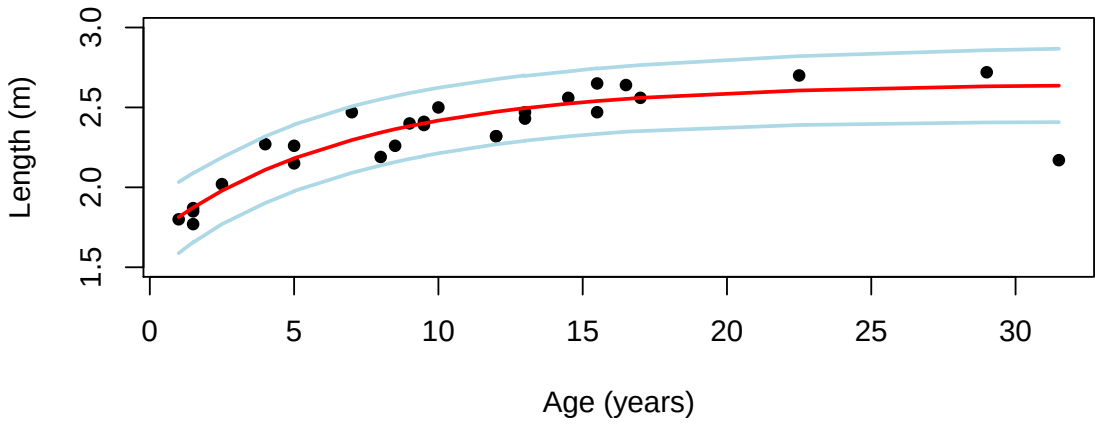
Predictions from robust model:

```
model.fit(dugongs_outl$Y, dugongs_outl$x, dugongs.A5.coda, "Y.pred",
          pch=20, lwd=2, xlab="Age (years)", ylab="Length (m)",
          ylim=c(1.5,3))
```



Predictions from Normal model

```
model.fit(dugongs_out1$Y, dugongs_out1$x, dugongs.A1.coda, "Y.pred",
          pch=20, lwd=2, xlab="Age (years)", ylab="Length (m)",
          ylim=c(1.5,3))
```



6. Try fitting a linear regression model: `mu[i] <- alpha + beta * x[i]` (which is rather inappropriate) and use DIC to compare the fit of this model to the non-linear model. Note that there will be no `gamma` parameter in the linear model, but you can just leave it in the code as a prior distribution to avoid having to edit the initial values to remove the `gamma`'s.

   To monitor DIC, set `DIC=TRUE` in the `bugs()` function. Estimates of `pD` and `DIC` are recorded as values of the `bugs` R object.

   The new model file should look like:

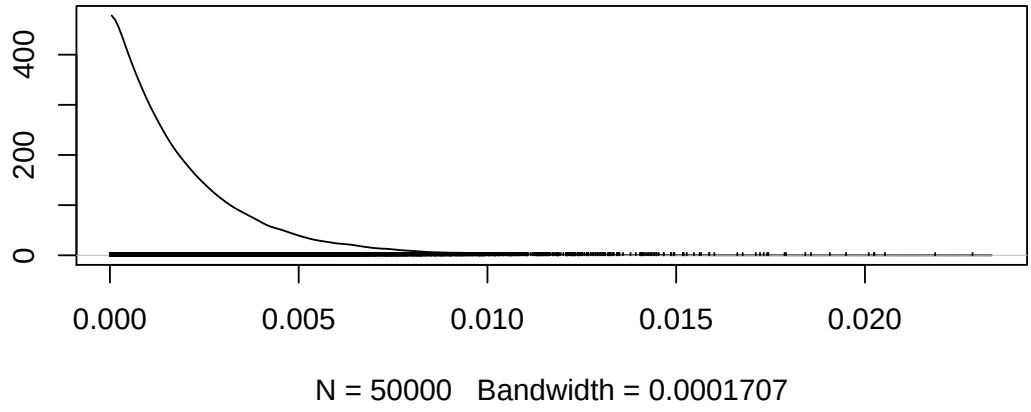
```
model {
for (i in 1:N){
  Y[i] ~ dnorm(mu[i], tau)
  mu[i] <- alpha - beta * x[i]

  res[i] <- (Y[i] - mu[i]) / sigma
  p.res[i] <- phi(res[i])
  Y.pred[i] ~ dnorm(mu[i], tau)
  P.pred[i]  <-  step(Y[i] - y.pred[i])
}
alpha ~ dunif(0, 100)
beta ~ dunif(0, 100)

# leave in code as prior - won't affect rest of model
gamma ~ dunif(0, 1.0)

tau ~ dgamma(0.001, 0.001)

# standard deviation of Normal errors
sigma <- 1/sqrt(tau)
}
```

```
dugongs_model_A6_file <- normalizePath("solutions/practical4/dugongs-model-A6.txt")
dugongs_in2_A6 <- list(alpha = 10, beta = 5, tau = 2, gamma = 0.2)
parameters.to.save <- c("alpha","beta","gamma","mu","sigma","res","P.res",
                        "P.pred","Y.pred")

dugongs.A6.sim <- bugs(model=dugongs_model_A6_file,
                       data=dugongs_outl,
                       inits=list(dugongs_in1, dugongs_in2_A6),
                       n.chains=2,
                       n.burnin=5000,
                       n.iter=55000,
                       n.thin=1,
                       parameters.to.save=parameters.to.save,
                       DIC=TRUE)
dugongs.A6.coda <- as.mcmc.list(dugongs.A6.sim)
dugongs.A6.sim[c("pD","DIC")]
```

```
## $pD
## [1] 2.044
##
## $DIC
## [1] 11.544
```

Notice that the estimate of pD appears to be too small here - there are actually 3 parameters in this model (alpha, beta and tau), but pD is close to 2. This is mainly because the posterior distribution of beta is very skewed, and using the posterior mean of beta as the the 'plug-in' estimate to calculate Dhat may not be very appropriate (see lecture notes for further discussion).

```
densplot(dugongs.A6.coda[,"beta"])
```



Compare with DIC from non-linear model with Normal errors:

Here there are actually 4 parameters in the model (alpha, beta, gamma and tau) and pD gives a close approximation. The non-linear model is also substantially superior to the linear model, since it has a much smaller DIC value.

B. Beetles

1. Use DIC to compare the 3 alternative link functions for the beetles data from Practical 3.

```
parameters.to.save <- c("alpha","beta","p")

beetles.B1.sim <- bugs(model="beetles-model.txt",
                       data=beetles,
                       inits=list(beetles_in1,beetles_in2),
                       n.chains=2,
                       n.burnin=10000,
                       n.iter=20000,
                       n.thin=1,
                       parameters.to.save=parameters.to.save,
                       DIC=TRUE)
```

```
beetles.B1.coda <- as.mcmc.list(beetles.B1.sim)
beetles.B1.sim[c("pD","DIC")]
```

```
## $pD
## [1] 1.979
##
## $DIC
## [1] 41.401
```

cloglog link

```
beetles_model_B3a_file <- normalizePath("solutions/practical3/beetles-model-B3a.txt")
beetles_in1_B3a <- list(alpha=0, beta=0)
beetles_in2_B3a <- list(alpha=1, beta=1)
parameters.to.save <- c("alpha","beta","p")

beetles.B3a.sim <- bugs(model=beetles_model_B3a_file,
                        data=beetles,
                        inits=list(beetles_in1_B3a,beetles_in2_B3a),
                        n.chains=2,
                        n.burnin=10000,
                        n.iter=25000,
                        n.thin=1,
                        parameters.to.save=parameters.to.save,
                        DIC=TRUE)
beetles.B3a.coda <- as.mcmc.list(beetles.B3a.sim)
beetles.B3a.sim[c("pD","DIC")]
```

```
## $pD
## [1] 1.983
##
## $DIC
## [1] 33.618
```

Probit link

```
beetles_model_B3b_file <- normalizePath("solutions/practical3/beetles-model-B3b.txt")
beetles_in1_B3b <- list(alpha=0, beta=0)
beetles_in2_B3b <- list(alpha=1, beta=1)
parameters.to.save <- c("alpha","beta","p")

beetles.B3b.sim <- bugs(model=beetles_model_B3b_file,
                        data=beetles,
                        inits=list(beetles_in1_B3b,beetles_in2_B3b),
                        n.chains=2,
                        n.burnin=10000,
```

```
                        n.iter=25000,
                        n.thin=1,
                        parameters.to.save=parameters.to.save,
                        DIC=TRUE)
beetles.B3b.coda <- as.mcmc.list(beetles.B3b.sim)
beetles.B3b.sim[c("pD","DIC")]
```

```
## $pD
## [1] 1.991
##
## $DIC
## [1] 40.303
```

DIC shows clear support for cloglog link model over logit and probit

Practical 5: Hierarchical models

THM data

A simulated set of data (slightly different from that used in the lecture notes) for the THM example can be found in R list `thm`. The code for fitting the basic hierarchical model is in the file `thm-model.txt`. Some initial values are in the R lists `thm_in1` and `thm_in2`.

1. Run the basic model, setting monitors on the zone mean THM concentrations `theta` and on the VPC, random effects mean and variance and the residual error variance. Also monitor the DIC. Produce box plots of the posterior distribution of the zone mean THM levels, and make a note of the DIC.

```
parameters.to.save <- c("mu","sigma2","psi2","vpc","theta")

thm.A1.sim <- bugs(model="thm-model.txt",
                   data=thm,
                   inits=list(thm_in1,thm_in2),
                   n.chains=2,
                   n.burnin=5000,
                   n.iter=15000,
                   n.thin=1,
                   parameters.to.save=parameters.to.save,
                   DIC=TRUE)
thm.A1.coda <- as.mcmc.list(thm.A1.sim)
thm.A1.sim[c("pD","DIC")]
```

```
## $pD
## [1] 57.374
##
## $DIC
## [1] 807.241
```

```
summary(thm.A1.coda[,c("mu","sigma2","psi2","vpc")])
```

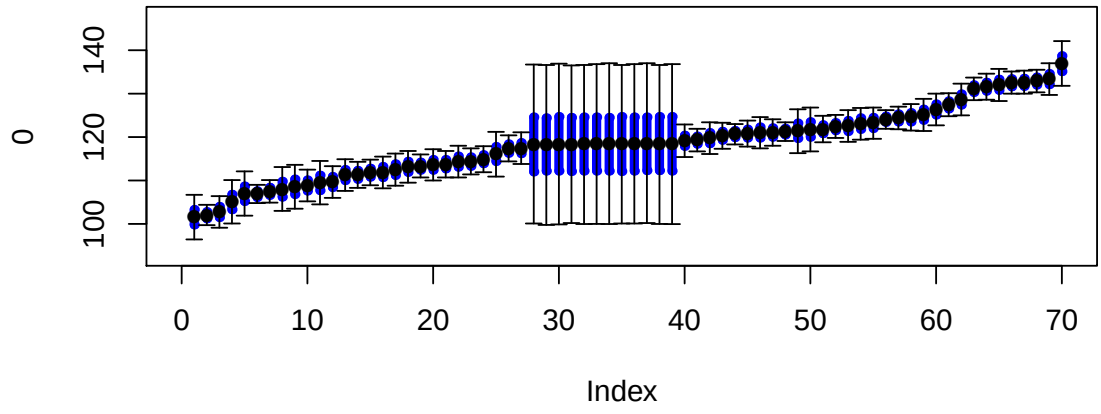
```
##
## Iterations = 5001:15000
## Thinning interval = 1
## Number of chains = 2
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##            Mean       SD  Naive SE Time-series SE
```

```
## mu     118.4086  1.23094 0.0087041       0.009040
## sigma2   7.2441  1.05369 0.0074507       0.011074
## psi2    85.4753 17.23962 0.1219025       0.140108
## vpc      0.9193  0.01841 0.0001302       0.000167
##
## 2. Quantiles for each variable:
##
##             2.5%      25%      50%      75%   97.5%
## mu     116.0000 117.6000 118.4000 119.2000 120.800
## sigma2   5.4640   6.4960   7.1420   7.8840   9.571
## psi2    57.8798  73.1700  83.3100  95.5900 124.700
## vpc      0.8787   0.9081   0.9211   0.9326   0.950
```

Note that VPC is high (around 92% of the variation is between zones), so hierarchically centred model should have better mixing and convergence properties.

Box plot of ranked zone mean THM levels (note the estimates for the 12 zones for which no measurements were available)

```
bugs.boxplot(thm.A1.coda, "theta",ordered=TRUE)
```



2. Edit the model code to allow the residual error variance to be zone specific, and assume a hierarchical prior distribution for the logarithms of these variances (see the N-of-1 example in the lecture notes, slides 6-39 to 6-44, or have a look at the solutions file). Think about how you should calculate the VPC for this model. Remember to edit the initial values files as appropriate.

   The model file should now look like:

```
model {
for (i in 1:Nobs) {
  thm[i] ~ dnorm(theta[zone[i]], tau[zone[i]])    # likelihood for observed data
}
for (z in 1:Nzone) {
  theta[z] ~ dnorm(mu, omega)    # zone-specific means (random effects)
  log.sigma2[z] ~ dnorm(nu, chi)    # zone-specific log variances (random effects)
  tau[z] <- 1/sigma2[z]
  sigma2[z] <- exp(log.sigma2[z])
}

# priors on random effects mean and SD
mu ~ dnorm(0, 0.000001)
# random effects SD (between-zone SD of mean THM)
psi ~ dunif(0, 1000)
psi2 <- pow(psi, 2)
omega <- 1 / psi2

nu ~ dunif(-100, 100)    # mean log variance
mean.var <- exp(nu)    # mean residual error variance
phi ~ dunif(0, 1000)
phi2 <- pow(phi, 2) # between-zone variance in log error variance
chi <- 1 / phi2

vpc <- psi2 / (psi2 + mean.var)    # average variance partition coefficient
}
```

3. Run the above model, produce box plots of the posterior distributions of the zone mean THM levels and the within zone residual variances, and report summary statistics for the VPC that you have calculated. Compare the DIC for this model with that of the original model—which model is preferred?

```
thm_model_A2_file <- normalizePath("solutions/practical5/thm-model-A2.txt")
thm_in1_A2 <- list(mu = 50, psi = 1, nu = 2, phi = 0.5)
thm_in2_A2 <- list(mu = 200, psi = 10, nu = 1, phi = 0.25)
parameters.to.save <- c("phi2","mean.var","mu","nu",
                        "psi2","sigma2","vpc","theta")

thm.A2.sim <- bugs(model=thm_model_A2_file,
                   data=thm,
                   inits=list(thm_in1_A2,thm_in2_A2),
                   n.chains=2,
                   n.burnin=5000,
                   n.iter=15000,
                   n.thin=100, # retain only every 100th iteration
```

```
                    parameters.to.save=parameters.to.save,
                    DIC=TRUE)
thm.A2.coda <- as.mcmc.list(thm.A2.sim)
thm.A2.sim[c("pD","DIC")]
```

```
## $pD
## [1] 70.291
##
## $DIC
## [1] 794.108
```

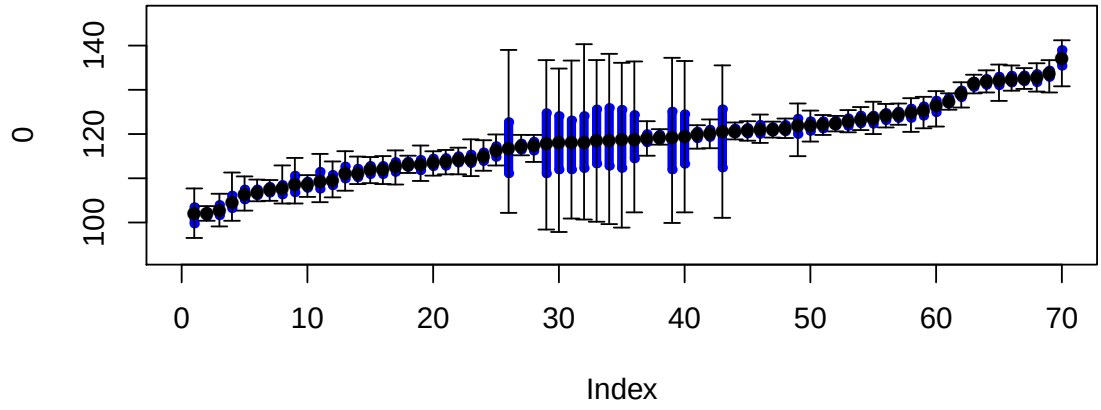
So DIC suggests hierarchical model for variances may be preferred over model assuming constant error variance. Note that the conclusions in WinBUGS/OpenBUGS do not match JAGS in this example: JAGS uses a different definition of pD that only matches the original definition in some settings (see The BUGS Book, p164-165).

```
summary(thm.A2.coda[,c("phi2","mean.var","mu","nu",
                       "psi2","vpc")])
```

```
##
## Iterations = 5001:14901
## Thinning interval = 100
## Number of chains = 2
## Sample size per chain = 100
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##               Mean       SD Naive SE Time-series SE
## phi2        0.2939  0.11792 0.008338       0.039251
## mean.var    5.4392  2.06997 0.146369       0.130264
## mu        118.4515  1.21059 0.085602       0.085699
## nu          1.6158  0.40205 0.028429       0.036589
## psi2       86.5052 16.26139 1.149854       1.063304
## vpc         0.9394  0.02459 0.001739       0.001245
##
## 2. Quantiles for each variable:
##
##                2.5%      25%      50%      75%    97.5%
## phi2        0.08502   0.2189   0.3004   0.3750   0.5225
## mean.var    2.86090   3.3890   5.3895   7.3742   8.4990
## mu        116.10000 117.5750 118.5000 119.4000 120.5025
## nu          1.05097   1.2208   1.6745   1.9980   2.1404
## psi2       59.80250  74.4000  84.5700  96.4550 119.9375
## vpc         0.88535   0.9215   0.9432   0.9619   0.9732
```

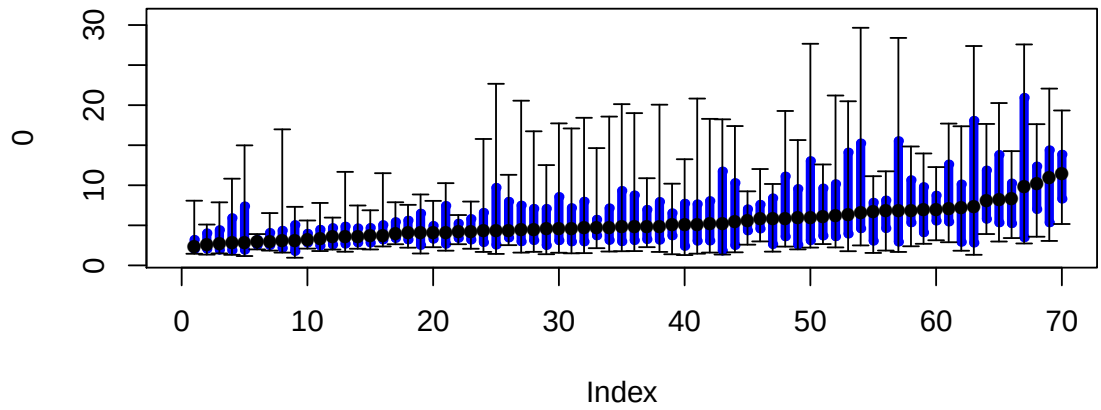
Box plot of ranked zone mean THM levels

```
bugs.boxplot(thm.A2.coda, "theta",ordered=TRUE)
```



Box plot of ranked zone-specific variances

```
bugs.boxplot(thm.A2.coda, "sigma2",ordered=TRUE)
```



4. The R list `thm_x` contains an additional covariate for each water zone, indicating the (standardised) average residence time (time water remains in the supply pipes between the source and the tap) for the tap water supply in each zone. Longer residence times often lead to greater fluctuations in THM levels in tap water. Edit your code to replace the random effects model for the zone-specific variances with a model that allows these variances to depend on residence time. Re-run the model and compare its fit with the previous models

using DIC. Which model is preferred? Is there evidence of an association between variability in THM levels and residence time in these data?

The model file should now look like:

```
model {
for (i in 1:Nobs){
thm[i] ~ dnorm(theta[zone[i]], tau[zone[i]])    # likelihood for observed data
}

for (z in 1:Nzone){
theta[z] ~ dnorm(mu, omega)    # zone-specific means (random effects)
log.sigma2[z] <- alpha + beta * x[z]    # zone-specific log variances
tau[z] <- 1/exp(log.sigma2[z])
}

# priors on random effects mean and SD
mu ~ dnorm(0, 0.000001)
# random effects SD (between-zone SD of mean THM)
psi ~ dunif(0, 1000)
psi2 <- pow(psi, 2)
omega <- 1 / psi2

alpha ~ dnorm(0, 0.00001)    # log variance for average residence time
beta ~ dnorm(0, 0.00001)    # increase in log variance for unit increase in residen
mean.var <- exp(alpha)    # residual variance for zone with average residence time

vpc <- psi2 / (psi2 + mean.var)    # average variance partition coefficient
}
```

```
thm_model_A4_file <- normalizePath("solutions/practical5/thm-x-model-A4.txt")
thm_in1_A4 <- list(mu = 50, psi = 1, alpha = 2, beta = 0.5)
thm_in2_A4 <- list(mu = 200, psi = 10, alpha = 1, beta = 0.1)
thm_x_all <- c(thm_x, thm) # append thm_x data to the original thm data
parameters.to.save <- c("beta","mean.var","mu","psi2","vpc","theta")

thm.A4.sim <- bugs(model=thm_model_A4_file,
                   data=thm_x_all,
                   inits=list(thm_in1_A4,thm_in2_A4),
                   n.chains=2,
                   n.burnin=10000,
                   n.iter=40000,
                   n.thin=1,
                   parameters.to.save=parameters.to.save,
```

```
                    DIC=TRUE)
thm.A4.coda <- as.mcmc.list(thm.A4.sim)
thm.A4.sim[c("pD","DIC")]
```

```
## $pD
## [1] 57.234
##
## $DIC
## [1] 806.398
```

So DIC suggests not much improvement in fit obtained by modelling variance as a function of residence time. Note that the conclusions in WinBUGS/OpenBUGS do not match JAGS in this example: JAGS uses a different definition of pD that only matches the original definition in some settings (see The BUGS Book, p164-165).

```
summary(thm.A4.coda[,c("beta","mean.var","mu","psi2","vpc")])
```

```
##
## Iterations = 10001:40000
## Thinning interval = 1
## Number of chains = 2
## Sample size per chain = 30000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##              Mean       SD  Naive SE Time-series SE
## beta       0.2625  0.20196 8.245e-04       0.033362
## mean.var   6.9314  0.90616 3.699e-03       0.093252
## mu       118.4126  1.24221 5.071e-03       0.005280
## psi2      85.4889 17.69661 7.225e-02       0.085967
## vpc        0.9224  0.01739 7.098e-05       0.000854
##
## 2. Quantiles for each variable:
##
##              2.5%      25%      50%      75%    97.5%
## beta      -0.1174   0.1175   0.2557   0.4052   0.6688
## mean.var   5.3000   6.2980   6.8750   7.5200   8.8310
## mu       116.0000 117.6000 118.4000 119.2000 120.8000
## psi2      57.2700  73.0300  83.3200  95.5900 126.4000
## vpc        0.8842   0.9117   0.9239   0.9348   0.9521
```

Coefficient for effect of residence time on log variance suggests an tendency for greater variability in zones with higher residence time, but the effect is not statistically significant.

Box plot of ranked zone mean THM levels

```
bugs.boxplot(thm.A4.coda, "theta",ordered=TRUE)
```