

Bayesian statistics with R

6. Bayesian analyses in R with the Jags software

Olivier Gimenez

March 2021

Bayes in practice

Software implementation (R compatible)

Oldies but goodies:

- WinBUGS, OpenBUGS: Where it all began.
- Jags: What we will use in this course.

Software implementation (R compatible)

Oldies but goodies:

- WinBUGS, OpenBUGS: Where it all began.
- Jags: What we will use in this course.

The new kids on the block:

- Nimble: What I'm going for these days.
- Stan: Entirely different algorithmic approach.
- Greta: Dunno anything about it.

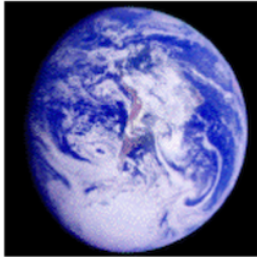
Introduction to JAGS (Just Another Gibbs Sampler)

Martyn Plummer



Real example

Impact of climatic conditions on white stork breeding success



mangl.at

Let's do a logistic regression on some White stork data

- Assess effects of temperature and rainfall on productivity.

Let's do a logistic regression on some White stork data

- Assess effects of temperature and rainfall on productivity.
- We have collected data.

Let's do a logistic regression on some White stork data

- Assess effects of temperature and rainfall on productivity.
- We have collected data.
- We need to build a model - write down the likelihood.

Let's do a logistic regression on some White stork data

- Assess effects of temperature and rainfall on productivity.
- We have collected data.
- We need to build a model - write down the likelihood.
- We need to specify priors for parameters.

Read in the data

```
nbchicks <- c(151,105,73,107,113,87,77,108,118,122,112,120,122,89,69,71,  
             53,41,53,31,35,14,18)  
  
nbpairs <- c(173,164,103,113,122,112,98,121,132,136,133,137,145,117,90,80,  
            67,54,58,39,42,23,23)  
  
temp <- c(15.1,13.3,15.3,13.3,14.6,15.6,13.1,13.1,15.0,11.7,15.3,14.4,14.4,  
          12.7,11.7,11.9,15.9,13.4,14.0,13.9,12.9,15.1,13.0)  
  
rain <- c(67,52,88,61,32,36,72,43,92,32,86,28,57,55,66,26,28,96,48,90,86,  
          78,87)  
  
datax <- list(N = 23, nbchicks = nbchicks, nbpairs = nbpairs,  
             temp = (temp - mean(temp))/sd(temp))
```

Write down the model

$\text{nbchicks}_i \sim \text{Binomial}(\text{nbpairs}_i, p_i)$ [likelihood]

$\text{logit}(p_i) = a + b_{\text{temp}} \text{temp}_i + b_{\text{rain}} \text{rain}_i$ [linear model]

$a \sim \text{Normal}(0, 1000)$ [prior for a]

$b_{\text{temp}} \sim \text{Normal}(0, 1000)$ [prior for b_{temp}]

$b_{\text{rain}} \sim \text{Normal}(0, 1000)$ [prior for b_{rain}]

Build the model

```
{  
# Likelihood  
  for( i in 1 : N){  
    nbchicks[i] ~ dbin(p[i],nbpairs[i])  
    logit(p[i]) <- a + b.temp * temp[i] + b.rain * rain[i]  
  }  
  
# ...
```

Specify priors

```
# Priors  
a ~ dnorm(0,0.001)  
b.temp ~ dnorm(0,0.001)  
b.rain ~ dnorm(0,0.001)  
}
```

Warning: Jags uses precision for Normal distributions (1 / variance)

You need to write everything in a file

```
model <-  
paste("  
model  
{  
  for( i in 1 : N)  
  {  
    nbchicks[i] ~ dbin(p[i],nbpairs[i])  
    logit(p[i]) <- a + b.temp * temp[i] + b.rain * rain[i]  
  }  
a ~ dnorm(0,0.001)  
b.temp ~ dnorm(0,0.001)  
b.rain ~ dnorm(0,0.001)  
}  
")
```

Alternatively, you may write a R function

```
logistic <- function() {  
  for( i in 1 : N)  
  {  
    nbchicks[i] ~ dbin(p[i],nbpairs[i])  
    logit(p[i]) <- a + b.temp * temp[i] + b.rain * rain[i]  
  }  
  
  # priors for regression parameters  
  a ~ dnorm(0,0.001)  
  b.temp ~ dnorm(0,0.001)  
  b.rain ~ dnorm(0,0.001)  
}
```


Let us specify a few additional things

```
# list of lists of initial values (one for each MCMC chain)
init1 <- list(a = -0.5, b.temp = -0.5, b.rain = -0.5)
init2 <- list(a = 0.5, b.temp = 0.5, b.rain = 0.5)
inits <- list(init1,init2)

# specify parameters that need to be estimated
parameters <- c("a","b.temp","b.rain")

# specify nb iterations for burn-in and final inference
nb.burnin <- 1000
nb.iterations <- 2000
```

Run Jags

```
# load R2jags
library(R2jags)
# run Jags
storks <- jags(data = datax,
               inits = inits,
               parameters.to.save = parameters,
               model.file = "code/logistic.txt",
               # model.file = logistic, # if a function was written
               n.chains = 2,
               n.iter = nb.iterations,
               n.burnin = nb.burnin)

storks
```

Inspect parameter estimates

```
#> Compiling model graph
#>   Resolving undeclared variables
#>   Allocating nodes
#> Graph information:
#>   Observed stochastic nodes: 23
#>   Unobserved stochastic nodes: 3
#>   Total graph size: 181
#>
#> Initializing model
#> Inference for Bugs model at "code/logistic.txt", fit using jags,
#> 2 chains, each with 2000 iterations (first 1000 discarded)
#> n.sims = 2000 iterations saved
```

#>	mu.vect	sd.vect	2.5%	25%	50%	75%	97.5%	Rhat	n.eff
#> a	1.548	0.084	1.429	1.518	1.553	1.589	1.669	1.196	2000
#> b.rain	-0.153	0.061	-0.264	-0.197	-0.153	-0.110	-0.045	1.018	2000
#> b.temp	0.028	0.062	-0.092	-0.011	0.026	0.067	0.157	1.007	230
#> deviance	206.443	31.697	201.780	202.774	203.886	205.682	213.467	1.089	2000

Your turn

- Run the stork analysis yourself.
- Does it seem like there is an effect of rainfall or temperature on breeding success?

Assess convergence

Reminder – MCMC Algorithm

- MCMC algorithms can be used to construct a Markov chain with a given stationary distribution (set to be the posterior distribution).

Reminder – MCMC Algorithm

- MCMC algorithms can be used to construct a Markov chain with a given stationary distribution (set to be the posterior distribution).
- For the MCMC algorithm, the posterior distribution is only needed to be known up to proportionality.

Reminder – MCMC Algorithm

- MCMC algorithms can be used to construct a Markov chain with a given stationary distribution (set to be the posterior distribution).
- For the MCMC algorithm, the posterior distribution is only needed to be known up to proportionality.
- Once the stationary distribution is reached we can regard the realisations of the chain as a (dependent) sample from the posterior distribution (and obtain Monte Carlo estimates).

Reminder – MCMC Algorithm

- MCMC algorithms can be used to construct a Markov chain with a given stationary distribution (set to be the posterior distribution).
- For the MCMC algorithm, the posterior distribution is only needed to be known up to proportionality.
- Once the stationary distribution is reached we can regard the realisations of the chain as a (dependent) sample from the posterior distribution (and obtain Monte Carlo estimates).
- We consider some important implementation issues.

MCMC – Proposal Distribution

- To implement a MCMC algorithm, we often need to specify a proposal distribution from which we generate candidate value then accept/reject.

MCMC – Proposal Distribution

- To implement a MCMC algorithm, we often need to specify a proposal distribution from which we generate candidate value then accept/reject.
- This typically involves
 - specifying a given distribution family (e.g. normal, uniform), and then,
 - setting the parameters of the given distribution.

MCMC – Proposal Distribution

- To implement a MCMC algorithm, we often need to specify a proposal distribution from which we generate candidate value then accept/reject.
- This typically involves
 - specifying a given distribution family (e.g. normal, uniform), and then,
 - setting the parameters of the given distribution.
- Although the exact distribution specified is essentially arbitrary – it will have a significant effect on the performance of the MCMC algorithm.

Why is the proposal distribution so important?

- If only small moves can be proposed, the acceptance probability is high, but it will take a long time to explore the posterior distribution.

Why is the proposal distribution so important?

- If only small moves can be proposed, the acceptance probability is high, but it will take a long time to explore the posterior distribution.
- Proposing large jumps has the potential to move further, but generally have smaller acceptance probabilities.

Why is the proposal distribution so important?

- If only small moves can be proposed, the acceptance probability is high, but it will take a long time to explore the posterior distribution.
- Proposing large jumps has the potential to move further, but generally have smaller acceptance probabilities.
- In order to balance the size of the proposed moves with the chance of accepting them the proposal variance is often tuned to obtain a mean acceptance probability of 20 – 40%.

Why is the proposal distribution so important?

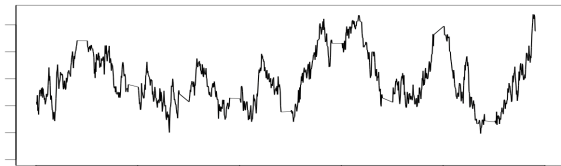
- If only small moves can be proposed, the acceptance probability is high, but it will take a long time to explore the posterior distribution.
- Proposing large jumps has the potential to move further, but generally have smaller acceptance probabilities.
- In order to balance the size of the proposed moves with the chance of accepting them the proposal variance is often tuned to obtain a mean acceptance probability of 20 – 40%.
- Automatic in Jags – ouf!

Why is the proposal distribution so important?

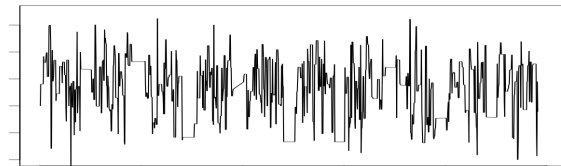
- If only small moves can be proposed, the acceptance probability is high, but it will take a long time to explore the posterior distribution.
- Proposing large jumps has the potential to move further, but generally have smaller acceptance probabilities.
- In order to balance the size of the proposed moves with the chance of accepting them the proposal variance is often tuned to obtain a mean acceptance probability of 20 – 40%.
- Automatic in Jags – ouf!
- The movement around the parameter space is often referred to as **mixing**.

Good/Bad Traces

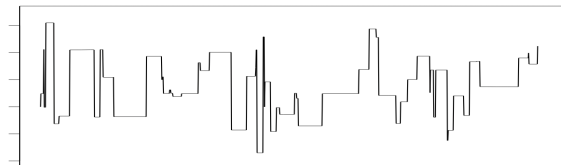
*Small
moves -
bad*



good



*Large
moves -
bad*



0

500

1000

Autocorrelation functions

- Traceplots of for small and big moves provide (relatively) high correlations (known as autocorrelations) between successive observations of the Markov chain.

Autocorrelation functions

- Traceplots of for small and big moves provide (relatively) high correlations (known as autocorrelations) between successive observations of the Markov chain.
- Strongly correlated observations require large sample sizes and therefore longer simulations.

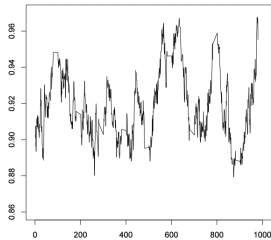
Autocorrelation functions

- Traceplots of for small and big moves provide (relatively) high correlations (known as autocorrelations) between successive observations of the Markov chain.
- Strongly correlated observations require large sample sizes and therefore longer simulations.
- Autocorrelation function (ACF) plots are a convenient way of displaying the strength of autocorrelation in the given sample values.

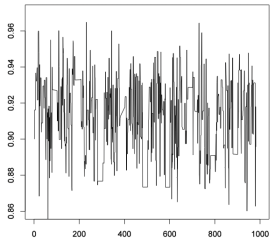
Autocorrelation functions

- Traceplots of for small and big moves provide (relatively) high correlations (known as autocorrelations) between successive observations of the Markov chain.
- Strongly correlated observations require large sample sizes and therefore longer simulations.
- Autocorrelation function (ACF) plots are a convenient way of displaying the strength of autocorrelation in the given sample values.
- ACF plots provide the autocorrelation between successively sampled values separated by k iterations, referred to as lag, (i.e. $\text{cor}(\theta_t, \theta_{t+k})$) for increasing values of k .

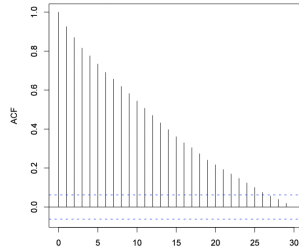
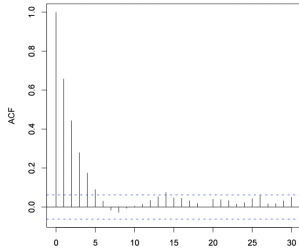
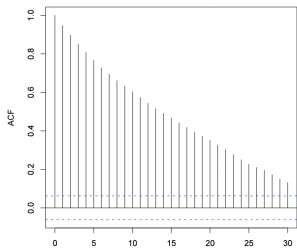
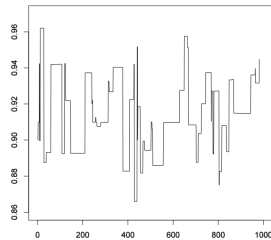
Small moves



OK

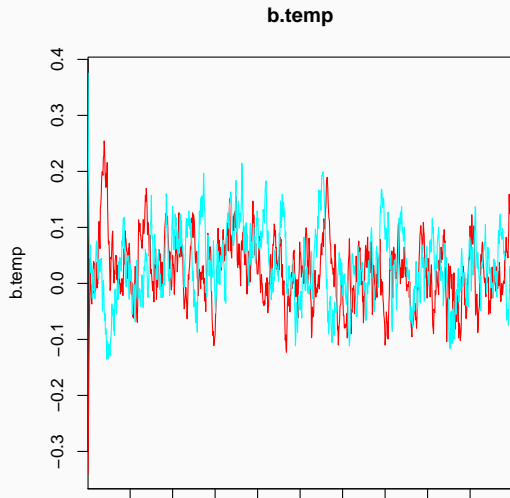
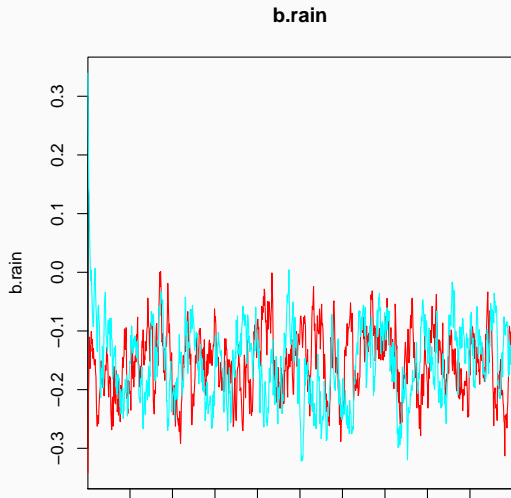


Big moves



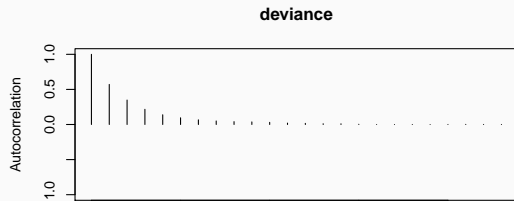
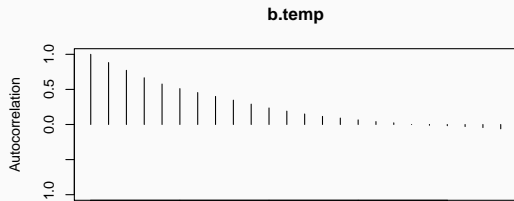
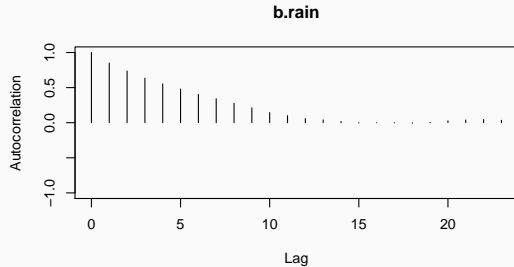
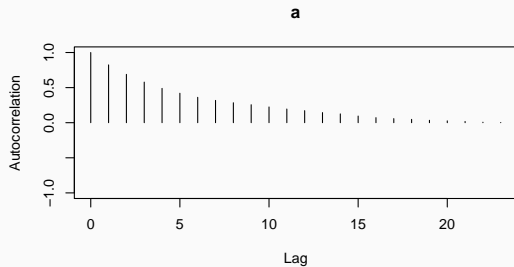
Traceplots for the storks

```
traceplot(storks,mfrow = c(1, 2), varname = c('b.rain','b.temp'), ask = FALSE)
```



Autocorrelation for the storks

```
autocorr.plot(as.mcmc(storks),ask = FALSE)
```



How do good chains behave?

- Converge to same target distribution: We need to think of the time required for convergence (realisations of the Markov chain have to be discarded before this is achieved).

How do good chains behave?

- Converge to same target distribution: We need to think of the time required for convergence (realisations of the Markov chain have to be discarded before this is achieved).
- Once there, explore efficiently: The post-convergence sample size required for suitable numerical summaries.

Convergence assessment

- Here, we are looking to determine how long it takes for the Markov chain to converge to the stationary distribution.

Convergence assessment

- Here, we are looking to determine how long it takes for the Markov chain to converge to the stationary distribution.
- In practice, we must discard observations from the start of the chain and just use observations from the chain once it has converged.

Convergence assessment

- Here, we are looking to determine how long it takes for the Markov chain to converge to the stationary distribution.
- In practice, we must discard observations from the start of the chain and just use observations from the chain once it has converged.
- The initial observations that we discard are referred to as the **burn-in**.

Convergence assessment

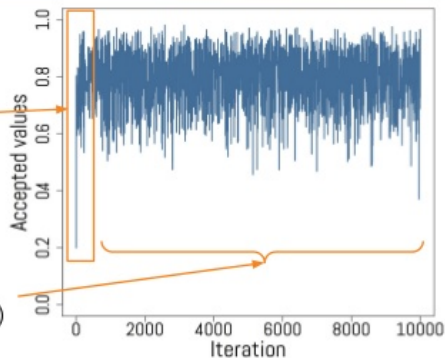
- Here, we are looking to determine how long it takes for the Markov chain to converge to the stationary distribution.
- In practice, we must discard observations from the start of the chain and just use observations from the chain once it has converged.
- The initial observations that we discard are referred to as the **burn-in**.
- The simplest method to determine the length of the burn-in period is to look at trace plots.

Burn-in (if simulations cheap, be conservative)

Discard initial guesses that are still far from optimum: the

BURN-IN

These numbers should be a good
sample of the Posterior $P(\phi \mid \text{data})$



Effective sample size n_{eff}

- How long of a chain is needed to produce stable estimates ?

Effective sample size n_{eff}

- How long of a chain is needed to produce stable estimates ?
- Most MCMC chains are strongly autocorrelated.

Effective sample size n_{eff}

- How long of a chain is needed to produce stable estimates ?
- Most MCMC chains are strongly autocorrelated.
- Successive steps are near each other, and are not independent.

Effective sample size `n.eff`

- How long of a chain is needed to produce stable estimates ?
- Most MCMC chains are strongly autocorrelated.
- Successive steps are near each other, and are not independent.
- The effective sample size (`n.eff`) measures chain length while taking into account the autocorrelation of the chain.
 - `n.eff` is less than the number of MCMC iterations.
 - Check the `n.eff` of every parameter of interest.
 - Check the `n.eff` of any interesting parameter combinations.

Effective sample size n_{eff}

- How long of a chain is needed to produce stable estimates ?
- Most MCMC chains are strongly autocorrelated.
- Successive steps are near each other, and are not independent.
- The effective sample size (n_{eff}) measures chain length while taking into account the autocorrelation of the chain.
 - n_{eff} is less than the number of MCMC iterations.
 - Check the n_{eff} of every parameter of interest.
 - Check the n_{eff} of any interesting parameter combinations.
- We need $n_{\text{eff}} \geq 100$ independent steps.

- Gelman-Rubin statistic \hat{R}

Potential scale reduction factor

- Gelman-Rubin statistic \hat{R}
- Measures the ratio of the total variability combining multiple chains (between-chain plus within-chain) to the within-chain variability. Asks the question is there a chain effect? Very much alike the F test in an ANOVA.

Potential scale reduction factor

- Gelman-Rubin statistic \hat{R}
- Measures the ratio of the total variability combining multiple chains (between-chain plus within-chain) to the within-chain variability. Asks the question is there a chain effect? Very much alike the F test in an ANOVA.
- Values near 1 indicates likely convergence, a value of ≤ 1.1 is considered acceptable.

Potential scale reduction factor

- Gelman-Rubin statistic \hat{R}
- Measures the ratio of the total variability combining multiple chains (between-chain plus within-chain) to the within-chain variability. Asks the question is there a chain effect? Very much alike the F test in an ANOVA.
- Values near 1 indicates likely convergence, a value of ≤ 1.1 is considered acceptable.
- Necessary condition, not sufficient; In other words, these diagnostics cannot tell you that you have converged for sure, only that you have not.

n.eff and \hat{R} for the storks

storks

```
#> Inference for Bugs model at "code/logistic.txt", fit using jags,  
#> 2 chains, each with 2000 iterations (first 1000 discarded)  
#> n.sims = 2000 iterations saved  
#>      mu.vect sd.vect  2.5%   25%   50%   75%  97.5% Rhat  
#> a      1.548  0.084  1.429  1.518  1.553  1.589  1.669 1.196  
#> b.rain -0.153  0.061 -0.264 -0.197 -0.153 -0.110 -0.045 1.018  
#> b.temp  0.028  0.062 -0.092 -0.011  0.026  0.067  0.157 1.007  
#> deviance 206.443 31.697 201.780 202.774 203.886 205.682 213.467 1.089  
#>  
#> For each parameter, n.eff is a crude measure of effective sample size,  
#> and Rhat is the potential scale reduction factor (at convergence, Rhat=  
#>  
#> DIC info (using the rule,  $pD = \text{var}(\text{deviance})/2$ )
```

To sum up

- Run multiple chains from arbitrary starting places (initial values).

To sum up

- Run multiple chains from arbitrary starting places (initial values).
- Assume convergence when all chains reach same regime.

To sum up

- Run multiple chains from arbitrary starting places (initial values).
- Assume convergence when all chains reach same regime.
- Discard initial burn-in phase.

To sum up

- Run multiple chains from arbitrary starting places (initial values).
- Assume convergence when all chains reach same regime.
- Discard initial burn-in phase.
- Check autocorrelation, effective sample size and \hat{R} .

What if you have issues of convergence?

- Increase burn-in, sample more.

What if you have issues of convergence?

- Increase burn-in, sample more.
- Use more informative priors.
- Pick better initial values (good guess).

What if you have issues of convergence?

- Increase burn-in, sample more.
- Use more informative priors.
- Pick better initial values (good guess).
- Reparameterize:
 - Standardize covariates.
 - Non-centering: $\alpha \sim N(0, \sigma)$ becomes $\alpha = z\sigma$ with $z \sim N(0, 1)$.

What if you have issues of convergence?

- Increase burn-in, sample more.
- Use more informative priors.
- Pick better initial values (good guess).
- Reparameterize:
 - Standardize covariates.
 - Non-centering: $\alpha \sim N(0, \sigma)$ becomes $\alpha = z\sigma$ with $z \sim N(0, 1)$.
- Something wrong with your model?
 - Start with a simpler model (remove complexities).
 - Use simulations.

What if you have issues of convergence?

- Increase burn-in, sample more.
- Use more informative priors.
- Pick better initial values (good guess).
- Reparameterize:
 - Standardize covariates.
 - Non-centering: $\alpha \sim N(0, \sigma)$ becomes $\alpha = z\sigma$ with $z \sim N(0, 1)$.
- Something wrong with your model?
 - Start with a simpler model (remove complexities).
 - Use simulations.
- Change your sampler. Upgrade to Nimble or Stan.

**MCMC makes you queens and kings
of the stats world**

Get all values sampled from posteriors

```
res <- as.mcmc(storks) # convert outputs in a list
res <- rbind(res[[1]],res[[2]]) # put two MCMC lists on top of each other
head(res)
#>               a      b.rain      b.temp  deviance
#> [1,] 0.04622874 -0.3417552 -0.33815288 1272.3221
#> [2,] 0.39929350 -0.2120148 -0.24532150  793.1198
#> [3,] 0.62334979 -0.1868884 -0.14196832  559.0869
#> [4,] 0.80827838 -0.1216267 -0.07715738  420.4606
#> [5,] 0.95639600 -0.1104892 -0.02060528  336.1302
#> [6,] 1.04438951 -0.1639800  0.04484738  294.1633
tail(res)
#>               a      b.rain      b.temp  deviance
#> [1995,] 1.448762 -0.1244035 -1.459287e-02 206.1755
#> [1996,] 1.457587 -0.1567102 -3.922608e-02 206.3416
```

Compute a posteriori $\Pr(\text{rain} < 0)$

```
# probability that the effect of rainfall is negative  
mean(res[, 'b.rain'] < 0)  
#> [1] 0.994
```

Compute a posteriori $\Pr(\text{temp} < 0)$

```
# probability that the effect of temperature is negative  
mean(res[, 'b.temp'] < 0)  
#> [1] 0.325
```


Get credible interval for the rain effect

```
quantile(res[, 'b.rain'], c(0.025, 0.975))  
#>           2.5%           97.5%  
#> -0.26365265 -0.04505914
```

Get credible interval for the temperature effect

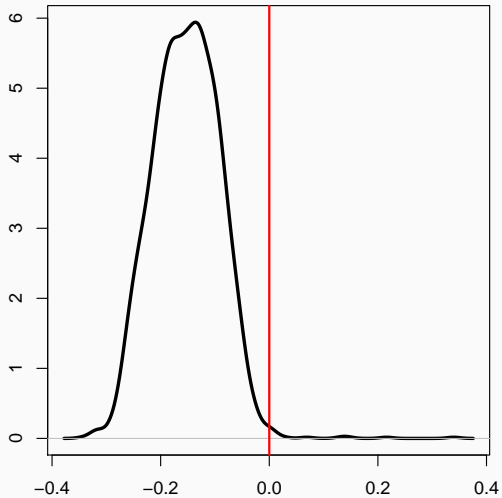
```
quantile(res[, 'b.temp'], c(0.025, 0.975))
```

```
#>           2.5%           97.5%
```

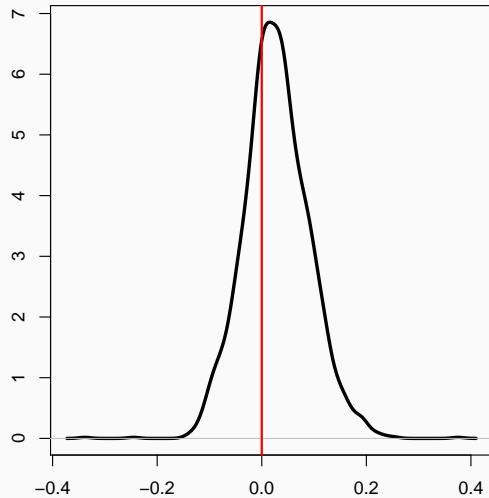
```
#> -0.09205657  0.15735156
```

Graphical summaries

Rainfall



Temperature



Your turn

A stupid question

- Get the posterior distribution of $b_{rain}^2 + \cos(b_{temp})$

Solution

- Evaluate the function for each MCMC iteration

```
stupid_pd <- res[, 'b.rain']^2 + cos(res[, 'b.temp'])  
head(stupid_pd)  
#> [1] 1.060166 1.015010 1.024867 1.011818 1.011996 1.025884
```

- Plot the distribution

```
plot(density(stupid_pd), xlab = '', main = '', lwd = 3)
```

