STA623 - Bayesian Data Analysis - Practical 5 (Solutions)

Marc Henrion

2025-09-26

Practical 5

Notation

- X, Y, Z - random variables
- x, y, z - measured / observed values
- $\bar{X}$, $\bar{Y}, \bar{Z}$ - sample mean estimators for X, Y, Z
- $\bar{x}$, $\bar{y}, \bar{z}$ - sample mean estimates of X, Y, Z
- $\hat{T}$, $\hat{t}$ - given a statistic T, estimator and estimate of T
- $P(A)$ - probability of an event A occuring
- $f_X(.)$, $f_Y(.), f_Z(.)$ - probability mass / density functions of X, Y, Z; sometimes $p_X(.)$ etc. rather than $f_X(.)$
- p(.) - used as a shorthand notation for pmfs / pdfs if the use of this is unambiguous (i.e. it is clear which is the random variable)
- $X \sim F$ - X distributed according to distribution function F
- $E[X]$, $E[Y]$, $E[Z]$, $E[T]$ - the expectation of X, Y, Z, T respectively

Exercise 1

Fit the model from Practical 3, Exercise 3 using `R` and `NIMBLE`.

Use this as the data from the sampling model:

$$y = (1, 3, 2, 3, 0, 2, 6, 4, 4, 1, 1, 3, 2, 3, 1, 1, 3, 0)$$

Inspect the trace plot and plot the posterior distribution.

Compute the posterior mean and the quantile-based 95% Bayesian confidence interval.

Exercise 1 (Solution)

First we need to write the `NIMBLE` model code:

```
library(nimble)
```

```
nimble version 1.3.0 is loaded.
For more information on NIMBLE and a User Manual,
please visit https://R-nimble.org.

Note for advanced users who have written their own MCMC samplers:
  As of version 0.13.0, NIMBLE's protocol for handling posterior
  predictive nodes has changed in a way that could affect user-defined
  samplers in some situations. Please see Section 15.5.1 of the User Manual.


Attaching package: 'nimble'

The following object is masked from 'package:stats':

    simulate

The following object is masked from 'package:base':

    declare
```

```
modelCode_Ex1<-nimbleCode(
  {
    # sampling model
    for(i in 1:N){
      y[i]~dpois(lambda)
    }

    # prior
    lambda~dgamma(5,2)
  }
)
```

This specifies the model.

Next we code up the data (for more complex data we would read these in from a data file).

```
y<-c(1,3,2,3,0,2,6,4,4,1,1,3,2,3,1,1,3,0)
  # 18 observations, y_i sum to 40
```

Now we need to fit this model using MCMC. We have two options with `NIMBLE`.

1. We can use a one-line `NIMBLE` MCMC invocation:

```
set.seed(123) # this sets a random seed; this is optional - it allows for reproducibility

parsPosterior_Ex1<-nimbleMCMC(
  code=modelCode_Ex1,
  constants=list(N=length(y)),
  data=list(y=y),
  monitors=c("lambda"),
  nchains=4,
  niter=1e4,
  nburnin=1e3,
  samplesAsCodaMCMC=TRUE # specify this to be able to handle the output easily with other MCI
)
```

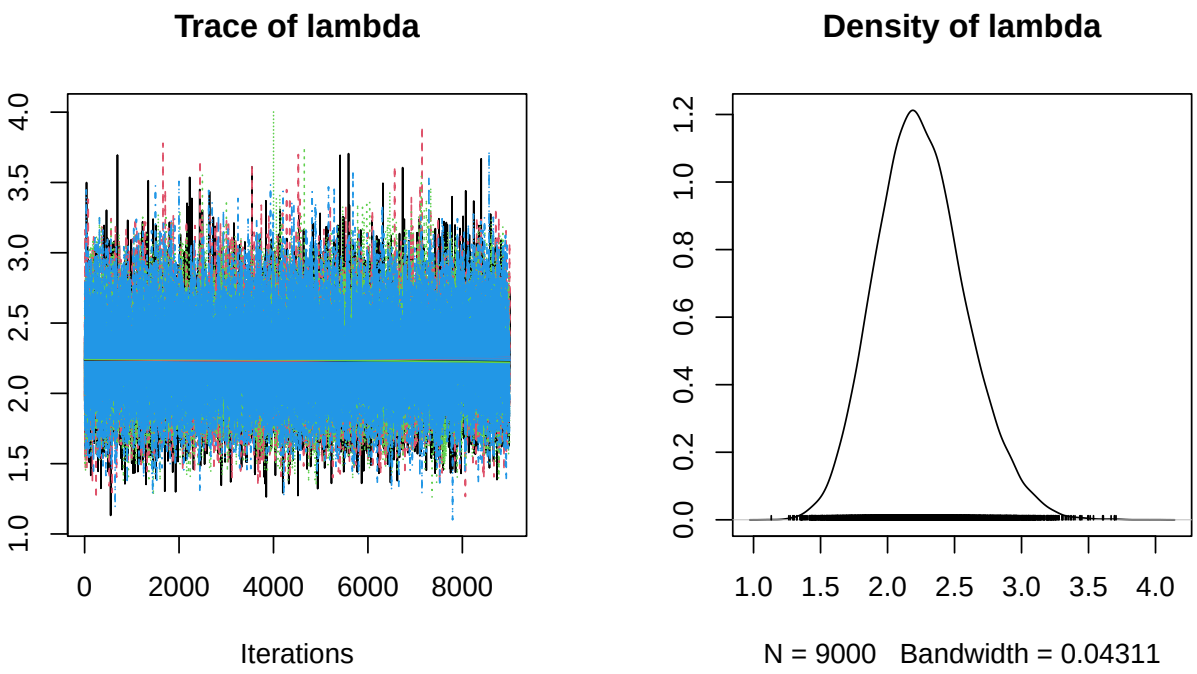
2. Or we can go manually through the different model and MCMC building steps for greater control:

```
# create model object
mod_Ex1<-nimbleModel(
  code=modelCode_Ex1,
  constants=list(N=length(y)),
  data=list(y=y)
)
## Defining model
## Building model
## Setting data and initial values
## Running calculate on model
##   [Note] Any error reports that follow may simply reflect missing values in model variabl
## Checking model sizes and dimensions
##   [Note] This model is not fully initialized. This is not an error.
##          To see which variables are not initialized, use model$initializeInfo().
##          For more information on model initialization, see help(modelInitialization).

# set up the MCMC
modConf_Ex1<-configureMCMC(
  model=mod_Ex1,
  monitors=c("lambda")
) # default random walk Metropolis sampler
## ===== Monitors =====
## thin = 1: lambda
## ===== Samplers =====
## conjugate sampler (1)
##   - lambda

modMCMC_Ex1<-buildMCMC(
  conf=modConf_Ex1
)

# compilation (optional - but next step will be much faster if you do this first)
compMCMCCode_Ex1<-compileNimble(mod_Ex1)
## Compiling
##   [Note] This may take a minute.
##   [Note] Use 'showCompilerOutput = TRUE' to see C++ compilation details.
compMCMCModel_Ex1<-compileNimble(modMCMC_Ex1,project=mod_Ex1)
## Compiling
##   [Note] This may take a minute.
##   [Note] Use 'showCompilerOutput = TRUE' to see C++ compilation details.

# run the MCMC (10,000 samples with 1,000 burn-in iterations)
```

```r
set.seed(123) # this sets a random seed; this is optional - it allows for reproducibility

parsPosterior_Ex1<-runMCMC(
  mcmc=compMCMCModel_Ex1, # specify modMCMC to run the uncompiled model if you skipped the co
  nchains=4,
  niter=1e4,
  nburnin=1e3,
  samplesAsCodaMCMC=TRUE
)
## running chain 1...
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
## running chain 2...
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
## running chain 3...
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
## running chain 4...
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
```

Next we can summarise the results. We use the `coda` package for this.

```r
library(coda)

# check trace plot and empirical posterior distribution
plot(parsPosterior_Ex1)
```




```
# posterior mean estimate
summary(parsPosterior_Ex1)$statistics["Mean"]
```

```
    Mean
2.251171
```

```
# posterior quantile based 95% credible interval
summary(parsPosterior_Ex1)$quantiles[c("2.5%","97.5%")]
```

```
    2.5%    97.5%
1.646866 2.951311
```

We could also use the `MCMCvis` package to summarise the output:

```
library(MCMCvis)

MCMCsummary(parsPosterior_Ex1)
```

```
           mean        sd     2.5%      50%    97.5% Rhat n.eff
lambda 2.251171 0.3337128 1.646866 2.232771 2.951311    1 36000
```

Exercise 2

Generate the following data

```
N<-100
x<-rnorm(N)
z<-2-4*x
p<-1/(1+exp(-z))
y<-rbinom(n=N,size=1,prob=p)

# I reformat the data and write it to a file on the hard drive, just so
# I can show you how to read data in and reformat for JAGS
df<-data.frame(
  x=x,
  y=y
)

write.csv(df,row.names=FALSE,quote=FALSE,file="Pract5Ex2Data.csv")
```

Use `R` and `NIMBLE` to fit a Bayesian logistic regression model to these data:

$$g(E[Y|X]) = \beta_0 + \beta_1 X$$

where $g(\pi) = \log(\pi/(1-\pi))$.

Compute the Gelman-Rubin convergence statistic and inspect trace plots and autocorrelations for the samples from the posterior distributions.

Compute the posterior mean, median, a 95% quantile-based confidence interval and a 95% highest posterior density confidence interval.

Compute the effective sample sizes for β_0, β_1.

Exercise 2 (solution)

First we load the data.

```
# read data
df<-read.csv("Pract5Ex2Data.csv")
```

Next we need to write the `NIMBLE` model code:

```
library(nimble)

modelCode_Ex2<-nimbleCode(
  {
    # logistic regression model
    for(i in 1:N){
      y[i]~dbern(p[i])
      logit(p[i])<-b0+b1*x[i]
    }

    # priors
    b0~dnorm(0,0.01) # try different priors; note default nimble parameterisation uses precis
    b1~dnorm(0,0.01) # try different priors; note default nimble parameterisation uses precis
  }
)
```

Then we proceed to specify the MCMC model.

```
# create model object
mod_Ex2<-nimbleModel(
  code=modelCode_Ex2,
  constants=list(N=nrow(df)),
  data=list(x=df$x,y=df$y)
)
```

```
Defining model

Building model

Setting data and initial values

Running calculate on model
  [Note] Any error reports that follow may simply reflect missing values in model variables.

Checking model sizes and dimensions

  [Note] This model is not fully initialized. This is not an error.
         To see which variables are not initialized, use model$initializeInfo().
         For more information on model initialization, see help(modelInitialization).
```

```
# set up the MCMC
modConf_Ex2<-configureMCMC(
  model=mod_Ex2,
  monitors=c("b0","b1")
) # default random walk Metropolis sampler -- which we will change below
```

```
===== Monitors =====
thin = 1: b0, b1
===== Samplers =====
RW sampler (2)
  - b0
  - b1
```

```
modConf_Ex2$removeSamplers(c("b0","b1")) # Remove default sampler if present
modConf_Ex2$addSampler(target = c("b0","b1"), type = 'polyagamma') # Polya-Gamma sampler for
```

```
modMCMC_Ex2<-buildMCMC(
  conf=modConf_Ex2
)

# compilation (optional - but next step will be much faster if you do this first)
compMCMCCode_Ex2<-compileNimble(mod_Ex2)
```

```
Compiling
  [Note] This may take a minute.
  [Note] Use 'showCompilerOutput = TRUE' to see C++ compilation details.
```

```
compMCMCModel_Ex2<-compileNimble(modMCMC_Ex2,project=mod_Ex2)
```

```
Compiling
  [Note] This may take a minute.
  [Note] Use 'showCompilerOutput = TRUE' to see C++ compilation details.
```

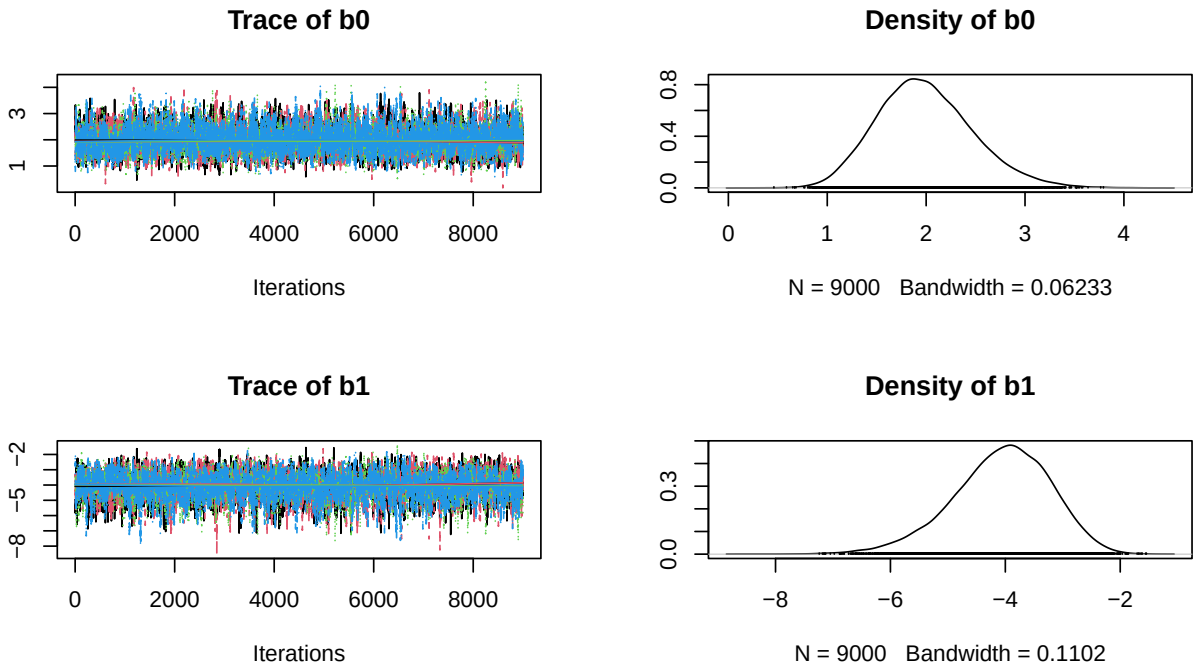
```
# run the MCMC (10,000 samples with 1,000 burn-in iterations)
set.seed(123) # this sets a random seed; this is optional - it allows for reproducibility

parsPosterior_Ex2<-runMCMC(
  mcmc=compMCMCModel_Ex2, # specify modMCMC to run the uncompiled model if you skipped the co
```

```
  nchains=4,
  niter=1e4,
  nburnin=1e3,
  samplesAsCodaMCMC=TRUE
)
```

```
running chain 1...

|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|

running chain 2...

|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|

running chain 3...

|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|

running chain 4...

|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

Finally we inspect and sumamrise the results:

```
# check trace plot and empirical posterior distribution
plot(parsPosterior_Ex2)
```



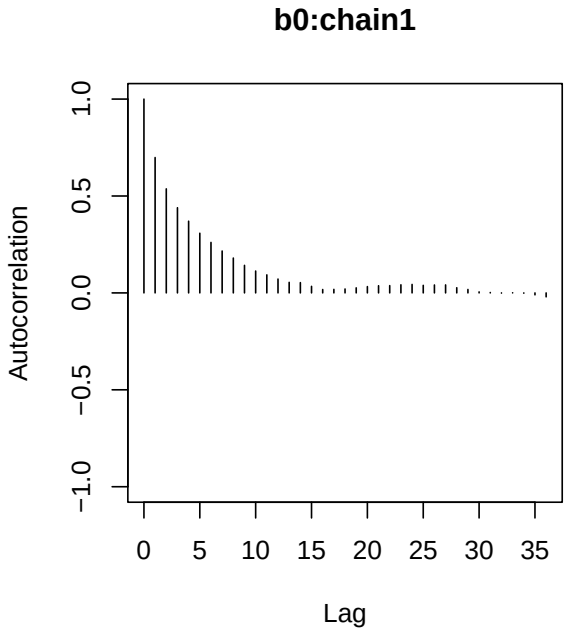

```
# model summary
MCMCsummary(parsPosterior_Ex2)
```

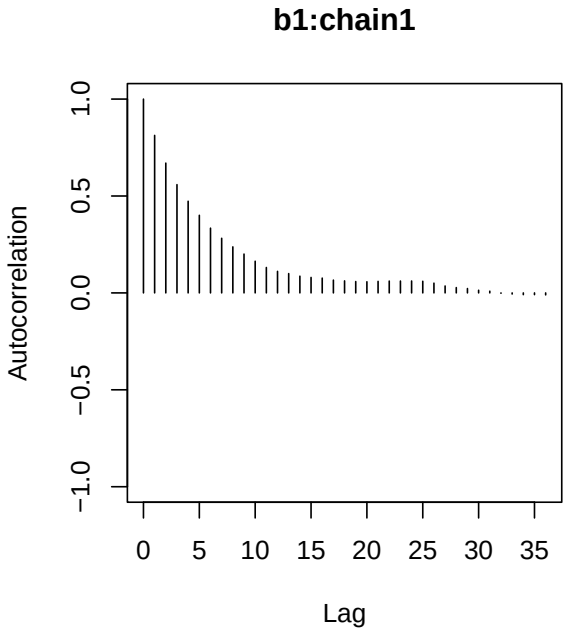
```
        mean        sd      2.5%       50%     97.5% Rhat n.eff
b0  1.989870 0.4824677  1.134974  1.959273  3.032168    1  3802
b1 -4.073642 0.8549181 -5.937692 -4.012479 -2.574906    1  2999
```

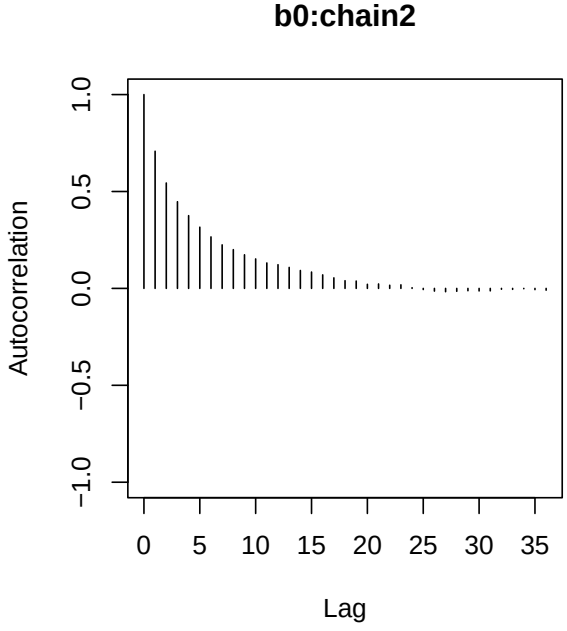
Trace plots, histograms look good and we get sensible potential scale reduction factors and effective sample sizes. So MCMC diagnostics look good.

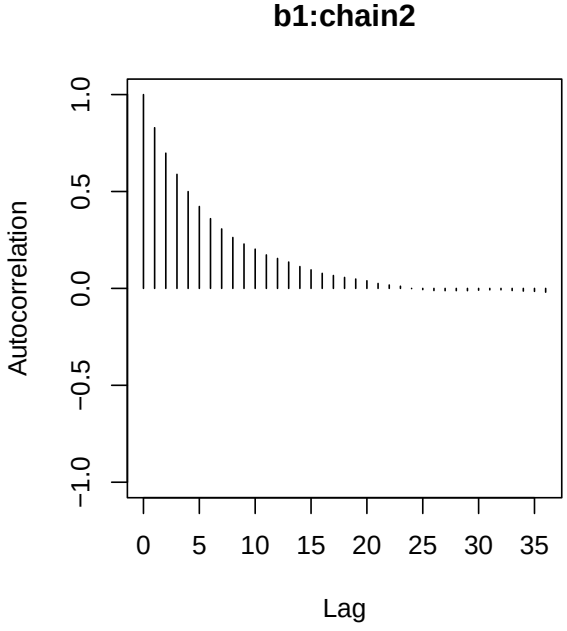
Regarding effective sample size and autocorrelations, we can actually inspect these directly, though the ESSs are a good summary. The `autocorr.plot()` functions produces an autocorreltion plot for each parameter and each chain - here we have 2 parameters, 4 chains, hence 8 graphs.

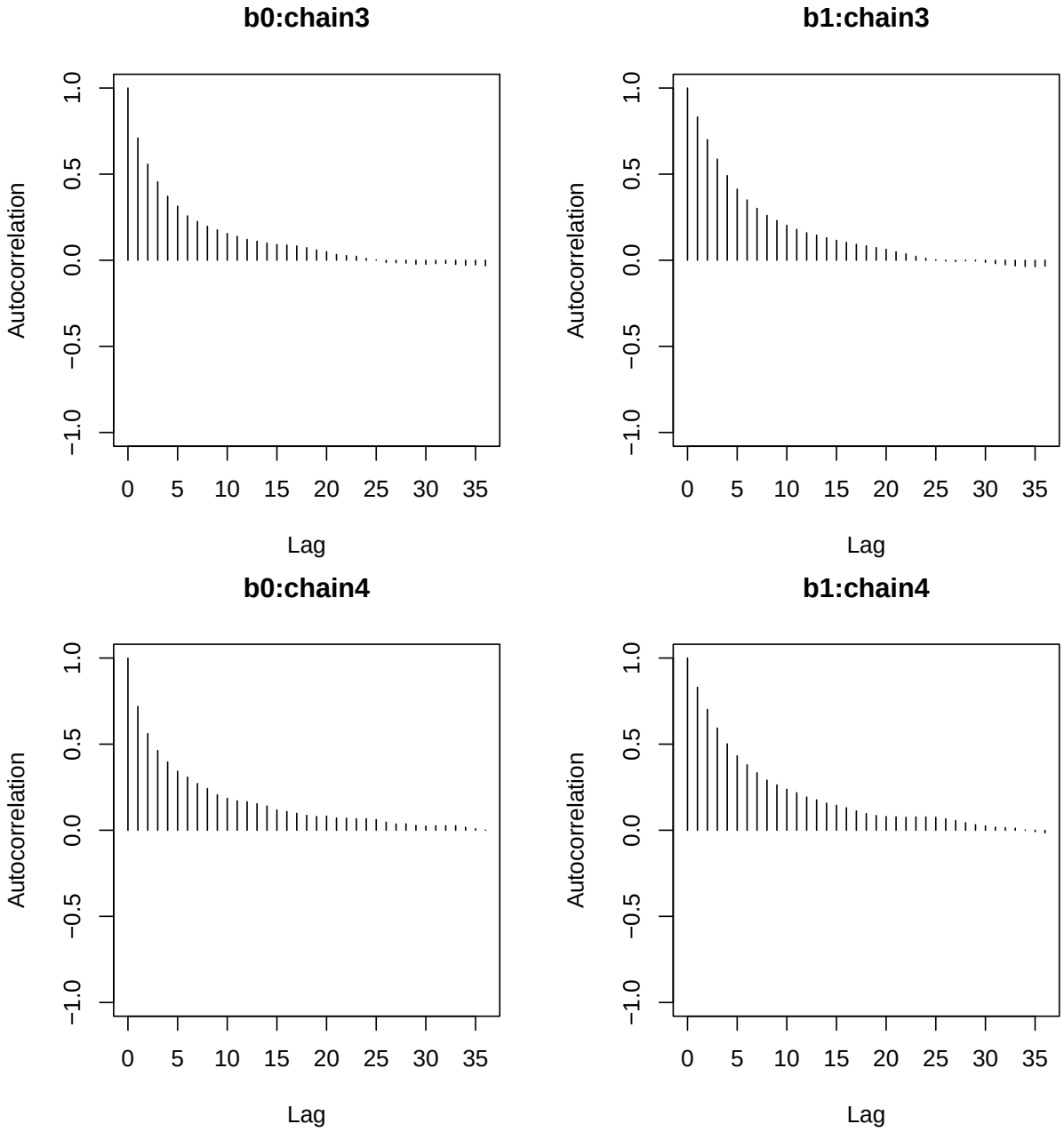
```
par(mfrow=c(4,2))
autocorr.plot(parsPosterior_Ex2,ask=FALSE)
```


b0:chain1
Autocorrelation
1.0
0.5
0.0
−0.5
−1.0
0
5
10
15
20
25
30
35
Lag


b1:chain1
Autocorrelation
1.0
0.5
0.0
−0.5
−1.0
0
5
10
15
20
25
30
35
Lag


b0:chain2
Autocorrelation
1.0
0.5
0.0
−0.5
−1.0
0
5
10
15
20
25
30
35
Lag


b1:chain2
Autocorrelation
1.0
0.5
0.0
−0.5
−1.0
0
5
10
15
20
25
30
35
Lag



We see that the autocorrelations drop rapidly and are negligible from iteration ~15 or so upwards.

In practice we would now also conduct posterior predictive checks to investigate that our model is suitable for the data. This is left as an exercise - but refer to the lecture notes for examples of this.

We now compute and report Bayesian point estimates together with 95% confidence intervals.

```
library(HDInterval)

tmp<-MCMCsummary(parsPosterior_Ex2)
tmp2<-hdi(parsPosterior_Ex2)

sumTab<-t(data.frame(
  posteriorMean=tmp[,"mean"],
  posteriorMedian=tmp[,"50%"],
  ci95_quantile_lower=tmp[,"2.5%"],
  ci95_quantile_upper=tmp[,"97.5%"],
  ci95_hdi_lower=tmp2["lower",rownames(tmp)],
  ci95_hdi_upper=tmp2["upper",rownames(tmp)]
))

rownames(sumTab)<-c("Posterior mean","Posterior median",
                    "95% CI lower (quantile)","95% CI upper (quantile)",
                    "95% CI lower (HDI)","95% CI upper (HDI)")

library(kableExtra)
```

```
Attaching package: 'kableExtra'

The following object is masked from 'package:dplyr':

    group_rows
```

```
sumTab %>%
  kable(digits=4,row.names=T) %>%
  kable_styling(full_width=F)
```

	b0	b1
Posterior mean	1.9899	-4.0736
Posterior median	1.9593	-4.0125
95% CI lower (quantile)	1.1350	-5.9377
95% CI upper (quantile)	3.0322	-2.5749
95% CI lower (HDI)	1.0825	-5.7712

95% CI upper (HDI)	2.9550	-2.4469

[end of STA623 BDA Practical 5]