A "Bayesian Analysis" Case Study using WinBUGS

Chiranjit Mukhopadhyay
Indian Institute of Science

1 Problem Formulation

Here we take up an example where we illustrate how to carry out Bayesian analysis of a real-life problem, on which some of you are already working. Here the problem is to define and estimate an elusive quantity called "true demand" of cars across different trims of a certain make and model. Conceptually this demand exists at different levels - starting from the most granulated level of the dealers, to DMA's, to different geographical regions of the country, to the last node of the hierarchy at the national level. Here we shall illustrate how to do it at the DMA level, though the methodology would essentially be the same for some other levels in the hierarchy as well.

Here we are interested in estimating the DMA level true demand for a trim of a certain make and model which has 6 trims. For this purpose we have observations on different variables, the structure of which will be described shortly, for 60 DMA's for 10 months. But before describing the data we first need to formally define the nature of the true demand, which is intrinsically assumed to be unobservable, and thus forms the parameter of the model. It is true that the sales, measured in terms of number of units, is reflective of the demand, but these sales figures by themselves are not true demands. Taking a cue from this basic assumption that sales reflect true demand, for $j = 1, 2, \ldots, 6$ and $i = 1, 2, \ldots, 60$ we define the true demand of the j-th trim in the i-th DMA as the quantity π_{ij} which gives the probability that a certain customer in the i-th DMA, who has already made up her mind about the make and the model, would want to buy a car belonging to the j-th trim. Thus $0 < \pi_{ij} < 1$ and $\sum_{j=1}^{6} \pi_{ij} = 1 \ \forall i = 1, 2, \ldots, 60$. In this case study we shall see how to estimate or more broadly draw inference about the parameters π_{ij}'s modeling the true demand.

Looking back at the definition of π_{ij}'s, it is obvious that the most critical observation that we shall need to draw inference about them would be the sales figures of the j-th trim for the i-th DMA. Thus for any given month if n_i cars of the given make and model have been sold in the i-th DMA, with Y_{ij} of them belonging to the j-th trim so that $\sum_{j=1}^{6} Y_{ij} = n_i$, then $\boldsymbol{Y}_i = (Y_{i1}, \ldots, Y_{i6}) \sim \text{Multinomial}(n_i; \pi_{i1}, \ldots, \pi_{i6})$. Note that although n_i's are random variables as well, we do not need a stochastic model for them in order to draw inference about the π_{ij}'s because π_{ij}'s give trim wise demand (which we are interested in) given the demand for the given make and model. The stochastic behavior of the n_i's would be of interest and would have been necessary to model if we were interested in estimating the demand of that particular make and the model instead.

Given only monthly observations on the $\boldsymbol{Y}_i$'s it is fairly straight-forward to draw inference about the π_{ij}'s. For instance, suppose in general there were J trims, I DMA's and we had observations for M months, with Y_{ijm} denoting the number of units sold in the j-th trim in

the i-th DMA in the m-th month, for $i = 1, 2, \ldots, I$, $j = 1, 2, \ldots, J$ and $m = 1, 2, \ldots, M$; and *apriori* $(\pi_{i1}, \ldots, \pi_{iJ}) \sim$Dirichlet$(\alpha_{i1}, \ldots, \alpha_{iJ})$. Then assuming that the observations were independent from month to month (a very bad assumption indeed, but we have to work with this wrong assumption for the lack of appropriate time series models for such discrete observations), from the structure of the Multinomial p.m.f. and Dirichlet p.d.f., it is a straight-forward exercise to show that the posterior of $(\pi_{i1}, \ldots, \pi_{ij}, \ldots, \pi_{iJ})$ in this case would have been Dirichlet$(\alpha_{i1} + Y_{i1.}, \ldots, \alpha_{ij} + Y_{ij.}, \ldots, \alpha_{iJ} + Y_{iJ.})$, where $Y_{ij.} = \sum_{m=1}^{M} Y_{ijm}$.

For the case study at hand however, we have additional information apart from the $\boldsymbol{Y}_{im}$'s, where $\boldsymbol{Y}_{im}=(Y_{i1m}, \ldots, Y_{iJm})$. One such information is the ADI values d_{ijm}'s computed using click-stream data on behavior of potential customers in the web, for the j-th trim in the i-th DMA, in the m-th month, expressed in terms of proportions so that $0 \leq d_{ijm} \leq 1$ and $\sum_{j=1}^{J} d_{ijm} = 1 \ \forall i = 1, \ldots, I$ and $m = 1, \ldots, M$. Intuitively these d_{ijm}'s are supposed to be a proxy for the parameters π_{ij}'s of interest. However these ADI values do not tally with the observed sales values $\boldsymbol{Y}_{im}$'s very well which in turn are supposed to be a good reflection of the true demand. Now we have a situation which is tailor-made for Bayesian analysis. We have some prior information about the π_{ij}'s expressed in terms of the d_{ijm}'s and then we have the concrete sales data, distribution of which directly depends on the π_{ij}'s as a Multinomial distribution. The Bayesian frame-work will now allow one to utilize both these streams of information. Thus with the model for the observed sales data already in place (*viz.* $\boldsymbol{Y}_{im} \sim$Multinomial$(\pi_{i1}, \ldots, \pi_{iJ}) \ \forall m = 1, \ldots, M$), now the task is to model the prior distribution of the π_{ij}'s using the d_{ijm}'s.

If we had no other information available, at this stage there would have been several choices for modeling the Dirichlet prior for the π_{ij}'s utilizing the d_{ijm}'s. One such would be a naive choice of $\alpha_{ij} = \overline{d}_{ij.} = \frac{1}{M} \sum_{m=1}^{M} d_{ijm}$. A slightly better approach would be that of a hierarchical Bayesian, where one puts a second stage prior on the Dirichlet hyperparameter $(\alpha_{i1}, \ldots, \alpha_{iJ})$ as let's say a Multivariate Normal distribution with mean vector $(\overline{d}_{i1.}, \ldots, \overline{d}_{iJ.})$ and $\frac{1}{M} \sum_{m=1}^{M} (d_{ijm} - \overline{d}_{ij.})(d_{ij'm} - \overline{d}_{ij'.})$ as the (j, j')-th element $(j, j' = 1, \ldots, J)$ of its variance-covariance matrix. Other approaches are also possible.

Here however we still have additional information in terms of the age of the sold cars. Thus let x_{ijm} denote the *average age of the sold cars* in the i-th DMA, j-th trim and m-th month. These x_{ijm}'s are computed as follows. Suppose at the beginning of the m-th month, i-th DMA starts with an inventory of v_{ijm} cars in the j-th trim, with the k-th such car being a_{ijmk} days old *i.e.* this car is lying in the parking lot without being sold for last a_{ijmk} days. Now suppose among these v_{ijm} cars, u_{ijm} of them get sold within that m-th month with the k-th one sold when it was a'_{ijmk} days old, $a_{ijmk} \leq a'_{ijmk} \leq a_{ijmk} + 30$. Then $x_{ijm} = \frac{1}{u_{ijm}} \left\{ \sum_{k=1}^{u_{ijm}} a'_{ijmk} + \sum_{k=1}^{v_{ijm} - u_{ijm}} (a_{ijmk} + 30) \right\}$. This gives the *average age of the sold cars*, because $\sum_{k=1}^{u_{ijm}} a'_{ijmk}$, the first term in the curly braces gives the total age of the cars that were actually sold in the m-th month (of which there are u_{ijm} of them), while $\sum_{k=1}^{v_{ijm} - u_{ijm}} (a_{ijmk} + 30)$ gives the total age of the cars that remained unsold till the end of the month (of which there are $v_{ijm} - u_{ijm}$ of them), and thus the total term within the curly braces gives the total age of the cars in the j-th trim for the i-th DMA during that m-th month. Now since the number of cars sold equals u_{ijm}, x_{ijm} defined as above yields the *average age of the sold cars*. Though the above formula for x_{ijm} might look a bit ad hoc, it

may be given a theoretical justification as follows. If one assumes that the number of days spent in the parking lot by a car before getting sold has an exponential distribution, which is a very reasonable model, and then attempts to estimate the mean of this exponential distribution using the ages of the cars, with the ones not sold regarded as right-censored, then one will come up with the above formula as the MLE of the mean.

Now we shall refine our model further with this additional information on the x_{ijm}'s. The x_{ijm}'s are used for refining both the model as well as the prior. Recall that our basic model is $\boldsymbol{Y}_{im} \sim$Multinomial$(\pi_{i1}, \ldots, \pi_{iJ})$ $\forall m = 1, \ldots, M$. Now since we are making the (quite unreasonable) assumption that the monthly sales figures are independent, in effect it is essentially same as assuming as if we had IM many independent DMA's and we have observation only for one month for each of them. Since we want to allow the π_{ij}'s to vary from DMA to DMA (expressed through their dependence on i), forgoing the time series structure, we now have $\boldsymbol{Y}_{im}$'s independent Multinomial$(\pi_{i1m}, \ldots, \pi_{iJm})$ as our basic model.

Now we want to utilize the x_{ijm}'s to further model the π_{ijm}'s, which is done as follows. Intuitively, if a π_{ijm} is small, that will lead to a large x_{ijm}. Thus conceptually though π_{ijm} is the cause and x_{ijm} is the effect (since we are modeling π_{ijm} as the true demand which basically controls both the x's and the Y's), if we can write π_{ijm} as a one-to-one function of x_{ijm}, then the (conceptual) causal relation between π_{ijm} and x_{ijm} may be viewed as the inverse of this function. The reason for taking this approach is that one has to be careful in modeling this relationship by keeping in mind that $0 \leq \pi_{ijm} \leq 1$ and $\sum_{j=1}^{J} \pi_{ijm} = 1$. One such one-to-one functional relationship between π_{ijm} and x_{ijm} ensuring the above constraint for the π_{ijm}'s may be expressed in terms of the logistic regression relationship $\pi_{ijm} = 1 \Big/ \left(1 + e^{-\beta_{j0} - \beta_{j1} x_{ijm}}\right)$, so that the causal relation between π_{ijm} and x_{ijm} is given by $x_{ijm} = -\beta_{j0} + (1/\beta_{j1}) \log(\pi_{ijm}/(1 - \pi_{ijm}))$, which may be interpreted as follows. If the log-odds of a customer demanding a car in trim-j increases, then the waiting time of selling the car measured in terms of its mean, decreases provided $\beta_{j1} < 0$. For instance, if $\beta_{j1} = -0.001$ and it is 1.1 times more likely for a customer to demand a car in trim j than other trims, then the expected waiting time to sell the cars in trim-j will decrease by 95 days. Thus finally our model for the sales observations $\boldsymbol{Y}_{im}$'s is as follows.

$$\begin{gathered} \boldsymbol{Y}_{im} \sim \text{Multinomial}(\pi_{i1m}, \ldots, \pi_{iJm}), \quad \boldsymbol{Y}_{11}, \ldots, \boldsymbol{Y}_{IM} \text{ independent} \\ \text{with } \pi_{ijm} = 1 \Big/ \left(1 + e^{-\beta_{j0} - \beta_{j1} x_{ijm}}\right) \end{gathered} \tag{1}$$

Above model is called the multinomial logit model. The model ensures that $0 < \pi_{ijm} < 1$. To ensure that $\sum_{j=1}^{J} \pi_{ijm} = 1$, we assume the above logit structure for the π_{ijm}'s for $j = 1, \ldots, J-1$ and then set $\pi_{iJm} = 1 - \sum_{j=1}^{J-1} \pi_{ijm}$. A typical multinomial logit model though is specified in a slightly different manner. We did not take the standard approach because of our ultimate interest in the π_{ijm}'s instead of the usual typical interests in the $\beta j \cdot$'s.

A couple of remarks regarding model (1) are in order. First note that we are assuming that the π_{ijm}'s are essentially dependent on j, expressed in terms of β_{j0} and β_{j1}. Thus these parameters essentially control the way the demand varies from one trim to another. The effect of a DMA on the demand on the other hand is assumed to be determined by an observable quantity, which at least on the surface appears to be reasonable. The second comment is about the assumption of the logit structure. It is an assumption and one could surely try

other models. For instance one can try the probit model given by $\pi_{ijm} = \Phi(\beta_{j0} + \beta_{j1} x_{ijm})$, where $\Phi(\cdot)$ is the standard Normal c.d.f.. Or one could try $\pi_{ijm} = F(\beta_{j0} + \beta_{j1} x_{ijm})$ for any continuous c.d.f. $F(\cdot)$. Here however for computational convenience we shall work with the logit model.

Now we discuss how to use the d_{ijm}'s, the ADI values, to come up with the prior distribution in presence of the additional information x_{ijm}'s. Model (1) has 2(J-1) parameters $((\beta_{10}, \beta_{11}), \ldots, (\beta_{J-1,0}, \beta_{J-1,1}))$, on which we are to put a prior. Since (β_{j0}, β_{j1}) are the parameters for the j-th trim we first assume that (β_{j0}, β_{j1}) and $(\beta_{j'0}, \beta_{j'1})$ are independent for $j \neq j'$. Next we assume that $(\beta_{j0}, \beta_{j1}) \sim N_2(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$ where $(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$ are the prior hyper-parameters. Thus for specifying the prior we need to specify this prior mean $\boldsymbol{\mu}_j$ and the prior variance-covariance matrix $\boldsymbol{\Sigma}_j$ of (β_{j0}, β_{j1}). We utilize the d_{ijm}'s to specify these prior hyper-parameters as follows.

Recall that the d_{ijm}'s are sort of the prior values of the π_{ijm}'s. Now in model (1), (β_{j0}, β_{j1})'s are related to the π_{ijm}'s through the logit relationship involving the x_{ijm}'s. Thus for specifying the prior for (β_{j0}, β_{j1}), it is natural to examine the same logit relationship of the d_{ijm}'s with the x_{ijm}'s. Thus we first propose the following simple linear regression model

$$\log\left(\frac{d_{ijm}}{1 - d_{ijm}}\right) = \beta_{j0} + \beta_{j1} x_{ijm} + \epsilon_{ijm} \quad \epsilon_{ijm}\text{'s i.i.d. } N(0, \sigma_j^2) \tag{2}$$

and then take the posterior distribution of (β_{j0}, β_{j1}) resulting from (2) with a non-informative prior on (β_{j0}, β_{j1}) as the prior for (β_{j0}, β_{j1}) of model (1). Since the posterior of (β_{j0}, β_{j1}) of model (2) with the non-informative prior $\pi(\beta_{j0}, \beta_{j1}, \sigma_j) \propto \frac{1}{\sigma_j}$ is approximately bivariate Normal with mean same as the MLE and variance-covariance matrix same as the inverse of the observed information matrix[1], we take $\boldsymbol{\mu}_j$ to be the MLE of (β_{j0}, β_{j1}) of model (2) and $\boldsymbol{\Sigma}_j$ as $\hat{\sigma}_j^2 \times \begin{bmatrix} IM & \sum_{i=1}^{I} \sum_{m=1}^{M} x_{ijm} \\ \sum_{i=1}^{I} \sum_{m=1}^{M} x_{ijm} & \sum_{i=1}^{I} \sum_{m=1}^{M} x_{ijm}^2 \end{bmatrix}^{-1}$, where $\hat{\sigma}_j^2$ is the UMVUE of σ_j^2 in model (2).

This completes the formulation of the problem. Note that the prior specification in terms of specifying the $(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$'s as discussed in the last paragraph can be carried out using any standard statistical software, and thus need not be coded in `WinBUGS` reducing its computational burden. These prior values are simply going to be read from an external data file in `WinBUGS`. However the posterior of (β_{j0}, β_{j1}) of model (1) with (β_{j0}, β_{j1}) *apriori* $N_2(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$ cannot be obtained in closed form and thus we have to code it and get it computed using `WinBUGS`.

[1]The exact marginal posterior of (β_{j0}, β_{j1}) is bivariate t with the same location and scale as that of the approximating bivariate Normal, with $IM - 2$ d.f.. But since in our case $IM = 600$, this bivariate t with $IM - 2$ d.f. is practically identical to the corresponding approximating bivariate Normal.

2 Data Analysis

We start our discussion with a description of the data set. We initially had 60 DMA's, 6 trims and data for 10 months, and for each of these we had observations on monthly sales, ADI values and the corresponding `x`-values. However there were quite a few `NA`'s for the `x`-values, and thus though `WinBUGS` and `R` can handle `NA`'s, to avoid unpredictable behavior, I first eliminated the records containing `NA` in at least one of its fields. Thus finally we had 401 records with $3 \times 6 = 18$ fields. Each record corresponds to a certain month of a DMA, and since we are not modeling the DMA and month separately, from now on we shall use the single subscript `i` (instead of the two subscripts i and m as in §1) to denote the `i`-th record, which is a combination of a certain DMA in a certain month with no `NA` in any its fields. Each record has 3 types of fields: `y`-field, `x`-field and `d`-field corresponding to the sales values, `x`-values and the ADI values, with each of these in turn containing 6 fields each corresponding to the 6 trims.

We shall first study the relationship between the `d[i,j]`'s and `x[i,j]`'s through (2) in `R` in order to come up with the hyperparameters $(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$'s. Then we use these in `WinBUGS` essentially to generate MCMC samples of posteriors of (β_{j0}, β_{j1}) in the model (1) with $(\beta_{j0}, \beta_{j1}) \sim N_2(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$ and output these as `CODA` files. Then these `CODA` files are again read into `R` for convergence diagnostics and the final Bayesian Analysis. We shall skip the basic `R` commands to get the initial data into the workspace, cleaning it, naming the variable etc. and shall only discuss the commands that are of data analytic interest.

2.1 Prior Specification

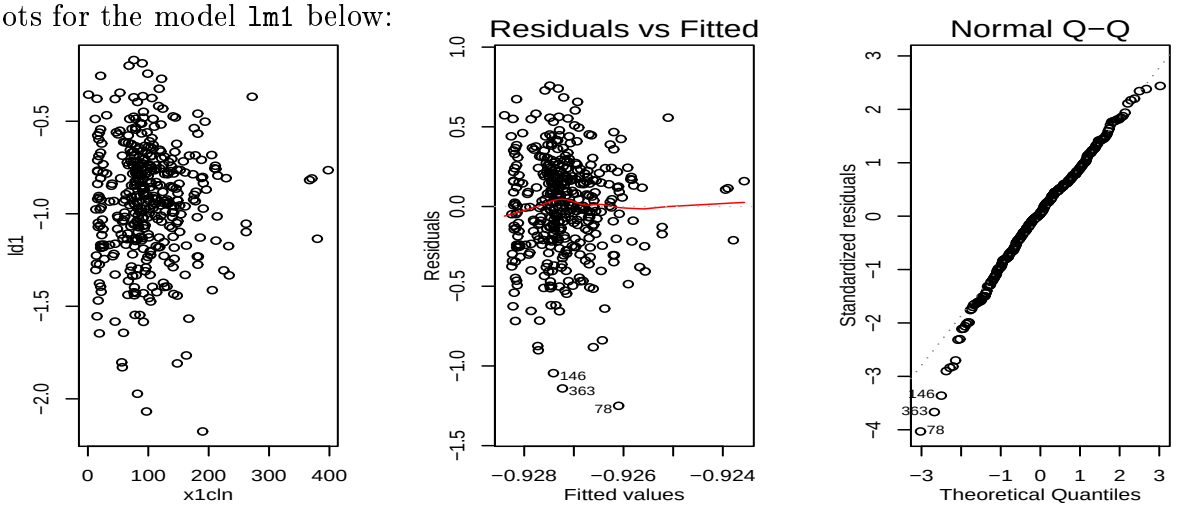
Logit of the `d[i,j]` values are stored in `ldj` *i.e.* `ld1` contains the logit of `d[i,1]`, `ld2` contains the logit of `d[i,2]` etc.. For obtaining the prior mean and dispersion of (β_{j0}, β_{j1}) using (2) we thus simply need to regress `ldj` on `xj` containing the `x[i,j]` values for 5 of the 6 `j`'s and pick-up the estimates of $(\beta_{j0}, \beta_{j1}, \sigma_j^2)$ and the so-called $\boldsymbol{X}'\boldsymbol{X}$ matrix from the output to prepare the mean vector and precision matrix of (β_{j0}, β_{j1}) to be supplied for the subsequent `WinBUGS` computation. We illustrate the `R` commands to do these for `j` $= 1$ as follows.

```
> lm1<-lm(ld1~x1cln,x=T)
> summary(lm1)

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -9.284e-01  3.141e-02 -29.557   <2e-16 ***
x1cln        1.216e-05  2.723e-04   0.045    0.964

Residual standard error: 0.3115 on 399 degrees of freedom
Multiple R-Squared: 4.999e-06,  Adjusted R-squared: -0.002501
F-statistic: 0.001995 on 1 and 399 DF,  p-value: 0.9644
```

```
> (1/(0.3115^2))*t(lm1$x)%*%lm1$x
            (Intercept)      x1cln
(Intercept)    4132.647   414212.8
x1cln        414212.800 55002991.3
```

`x1cln` contains the 401 cleaned (with no `NA`'s) values of `x1`. `lm()` is the `R` command to build a linear model of which (2) is a special case. `summary(lm1)` gives the summary statistics from which here we are only interested in the estimates of $(\beta_{10}, \beta_{11}, \sigma_1^2)$ given by (-0.9284,0.00001216, 0.3115^2). From these estimates we get $\boldsymbol{\mu}_1$=(-0.9284,0.00001216) as the prior mean of (β_{10}, β_{11}). Now note that `WinBUGS` requires a multivariate Normal distribution to be specified in terms of its mean vector and precision matrix instead of the usual dispersion matrix, which is nothing but the inverse of the precision matrix. The precision of (β_{10}, β_{11}) is given by $\frac{1}{\hat{\sigma}^2}\boldsymbol{X}'\boldsymbol{X}$. Of these $\hat{\sigma}$ has already been obtained as 0.3115. The $\boldsymbol{X}$ matrix of a regression model can be obtained as an `lm`-object`$x` in `R` , provided the `x=T` flag was on while creating the `lm`-object, as is the case for the `lm`-object `lm1` at hand. Thus `t(lm1$x)` gives $\boldsymbol{X}'$, the transpose of the $\boldsymbol{X}$ matrix and using the matrix multiplication operator `%*%`, `(1/(0.3115^2))*t(lm1$x)%*%lm1$x` gives us the precision matrix $\boldsymbol{\Sigma}_1^{-1}$ of (β_{10}, β_{11}). Now the values $\boldsymbol{\mu}_1$ =(-0.9284,0.00001216) and $\boldsymbol{\Sigma}_1^{-1} = \begin{bmatrix} 4132.647 & 414212.8 \\ 414212.800 & 55002991.3 \end{bmatrix}$ will be specified in a data file to be read by `WinBUGS` as the hyperparameters of the prior of (β_{10}, β_{11}) of model (1), which is assumed to be bivariate Normal.

It is customary to do a model checking for a regression analysis. For this we present three plots for the model `lm1` below:

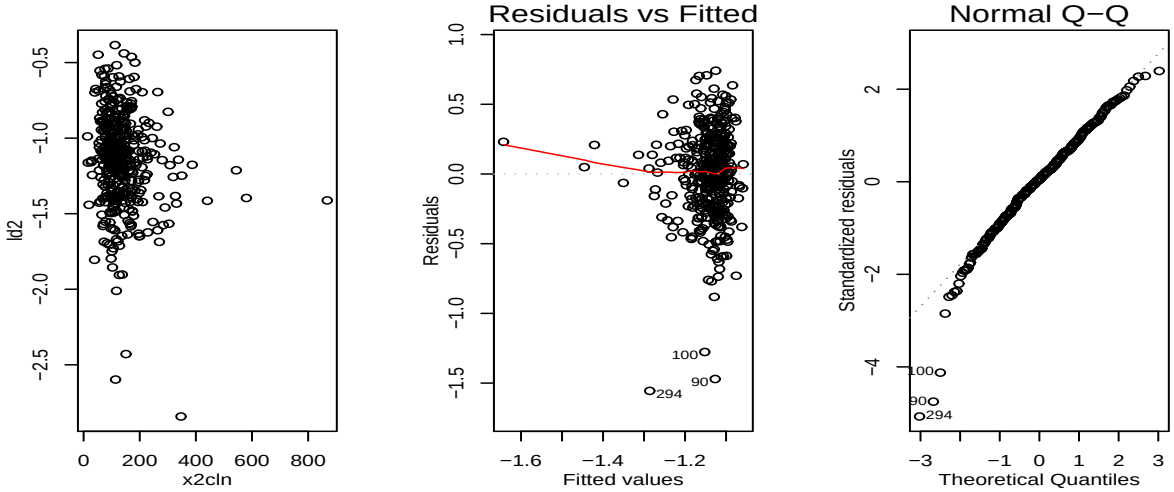


The first plot, called the scatter plot, simply plots the dependent variable `ld1` against the independent variable `x1cln`. We should expect to see a negative association between the two. Though the scatter plot gives such a loose impression, the estimate of β_{11} has turned out to be positive and more importantly it is not significant according to the frequentist p-value. Indeed the posterior probability of $\beta_{11} > 0$ is 0.518 and thus there is no reason to believe that $\beta_{11} < 0$ or even the other way round. What it means is that there is probably no association between `ld1` and `x1cln`. But this should not be construed as a hindrance for specifying the prior using this method, because the prior of β_{11} in this case is going to be fairly highly concentrated near 0, as has been depicted by the data.

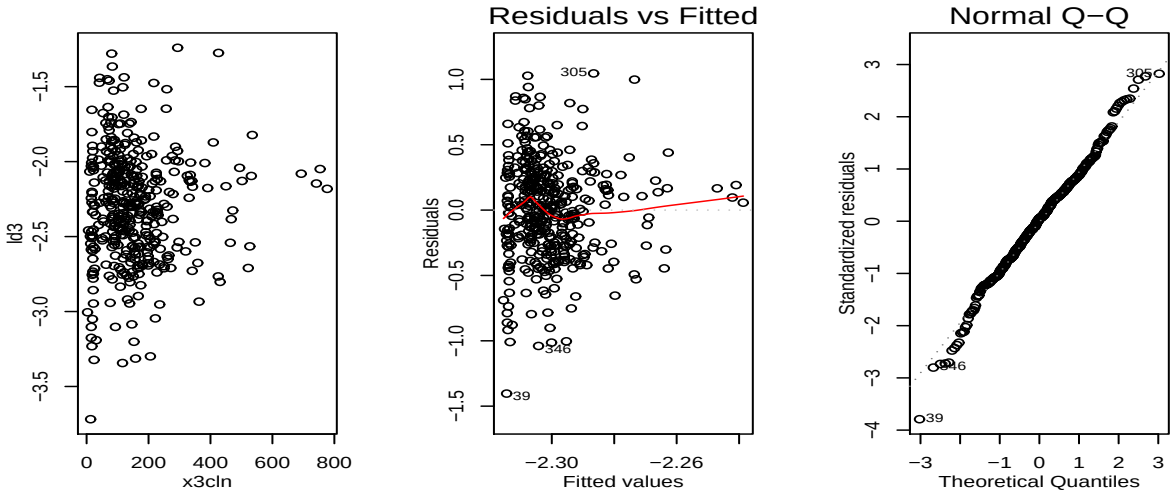
The second plot, called the residual plot checks whether there is still any pattern left in the data that has not been captured by the regression model and it also provides a check for the homoscedasticity assumption. Both of these seem to be loosely okay with the model. The third plot provides a graphical check for the Normality assumption. Since the plot appears to be more or less falling on a straight line, the normality assumption also appears to be approximately correct. Incidentally these two plots were generated using the command `plot.lm(lm1, which=1:2)` in `R` .

We finish this section by assorting the `R` outputs and plots for the remaining 4 cases. It should be remarked at this point that model (2) could not be fitted for the original trim 4 because of a near singular $\boldsymbol{X}$ matrix, and thus this became our natural choice of the trim for which we do not provide the β_j.'s and the corresponding π_{ij}'s are found via subtraction.

```
> lm2<-lm(ld2~x2cln,x=T)
> summary(lm2)

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.0486097  0.0314140 -33.380  < 2e-16 ***
x2cln       -0.0006844  0.0001999  -3.423 0.000683 ***

Residual standard error: 0.31 on 399 degrees of freedom
Multiple R-Squared: 0.02853,    Adjusted R-squared: 0.0261
F-statistic: 11.72 on 1 and 399 DF,  p-value: 0.000683

> (1/(0.31^2))*t(lm2$x)%*%lm2$x
            (Intercept)       x2cln
(Intercept)    4172.737    570509.9
x2cln        570509.886 103030031.2

> plot(x2cln,ld2)
> plot.lm(lm2,which=1:2)
```

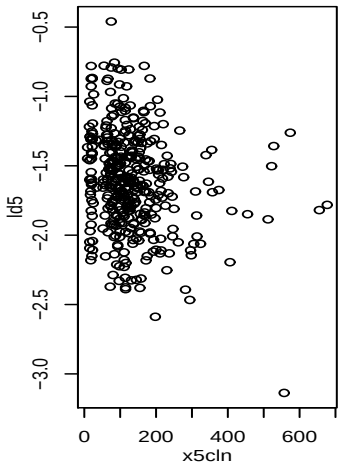


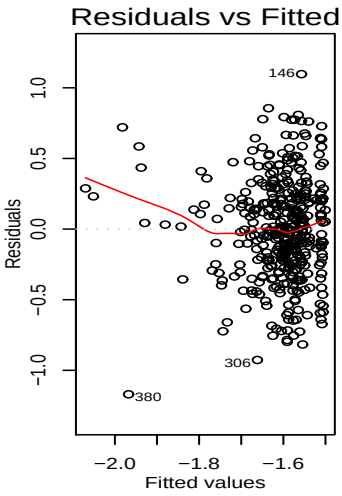
```
> lm3<-lm(ld3~x3cln,x=T)
> summary(lm3)

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -2.316e+00  3.038e-02 -76.229   <2e-16 ***
x3cln        9.921e-05  1.578e-04   0.629     0.53

Residual standard error: 0.3714 on 399 degrees of freedom
Multiple R-Squared: 0.0009902,  Adjusted R-squared: -0.001514
F-statistic: 0.3955 on 1 and 399 DF,  p-value: 0.5298

> (1/(0.3174^2))*t(lm3$x)%*%lm3$x
            (Intercept)       x3cln
(Intercept)    3980.435    607070.9
x3cln        607070.920 147609416.3

> plot(x3cln,ld3)
> plot.lm(lm3,which=1:2)
```

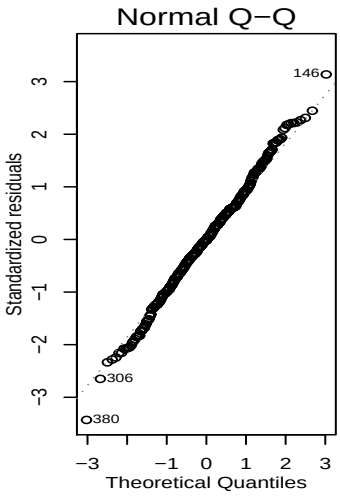



```
> lm5<-lm(ld5~x5cln,x=T)
> summary(lm5)

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.4929528  0.0297141  -50.24  < 2e-16 ***
x5cln       -0.0008519  0.0001836   -4.64 4.72e-06 ***

Residual standard error: 0.3503 on 399 degrees of freedom
Multiple R-Squared: 0.0512,     Adjusted R-squared: 0.04883
```

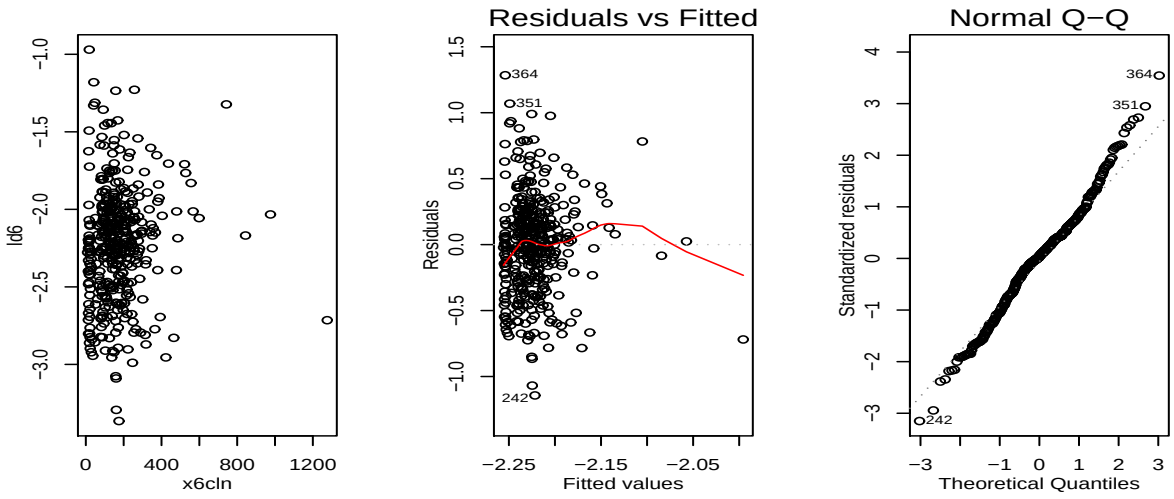
```
F-statistic: 21.53 on 1 and 399 DF,  p-value: 4.724e-06

> (1/(0.3503^2))*t(lm5$x)%*%lm5$x
            (Intercept)      x5cln
(Intercept)    3267.865   427552.5
x5cln        427552.453 85611435.9

> plot(x5cln,ld5)
> plot.lm(lm5,which=1:2)
```








```
> lm6<-lm(ld6~x6cln,x=T)
> summary(lm6)

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -2.2576785  0.0295908 -76.297   <2e-16 ***
x6cln        0.0002054  0.0001376   1.493    0.136

Residual standard error: 0.3633 on 399 degrees of freedom
Multiple R-Squared: 0.005555,   Adjusted R-squared: 0.003062
F-statistic: 2.229 on 1 and 399 DF,  p-value: 0.1363

> (1/(0.3633^2))*t(lm6$x)%*%lm6$x
            (Intercept)       x6cln
(Intercept)    3038.181    516225.5
x6cln        516225.513 140562801.4

> plot(x6cln,ld6)
> plot.lm(lm6,which=1:2)
```



Before proceeding to MCMC computation of the posterior of $\beta_{j.}$'s, it may be remarked that while the Normality assumption in all the five cases above look satisfactory, a couple of them might have some problem with homoscedasticity and none of them has problem with the assumed linear structure in (2). Also it is interesting to note that only when the estimated β_{j1} turned out to be negative, as per the expectation, as in the case of trims 3 and 5, they became significantly so. For the remaining 3 trims, the estimated values are positive and they are not significant meaning that these β_{j1}'s priors will be pretty much concentrated around 0, while for the other two they will be fairly heavily concentrated in the negative axis.

2.2 MCMC Computation

The model in (1) with independent bivariate Normal prior for the (β_{j0}, β_{j1})'s may be coded for `WinBUGS`' model specification file as follows:

```
model
{
 for(i in 1:401)
 {
  y[i,1:6]~dmulti(pi[i,1:6],n[i])
  for(j in 1:5)
  {
   logit(pi[i,j]) <- beta[j,1] + beta[j,2]*x[i,j]
  }
  pi[i,6] <- 1 - pi[i,1] - pi[i,2] - pi[i,3] - pi[i,4] - pi[i,5]
 }
 for(j in 1:5)
 {
  beta[j,1:2] ~ dmnorm(mu[j,1:2],T[j,1:2,1:2])
 }
}
```

Together with the above model file, WinBUGS is supplied with two data files - the first one containing $401 \times (1 + 6 + 6)$ n[], y[,] and x[,] values, with n[i]=$\sum_{j=1}^{6}$y[i,j]; and the second one containing the mu[,] and T[,,] values as obtained in §2.1. Given these, WinBUGS automatically figures out that, since no data has been specified for the stochastic beta[,] nodes, those must be our parameters of interest and starts simulating from their conditional distributions given the values specified for other stochastic and constant nodes (like mu[,] and T[,,]) as per the specified model. This is nothing but the posterior of our interest.

Typically one has to decide on three or four parameters while running an MCMC. In order to determine these parameters one typically first have to make a pilot run. Ideally one makes a pilot run of decent enough length, say in the order of 10,000 or so after discarding some initial (ad-hoc number of) observations. Then run a Raftery-Lewis diagnostics on this pilot chain to get an idea about the optimal values of the parameters that one should use for the MCMC run.

Here to minimize effort, I have taken a mixed approach, wherein part of the final sample itself is obtained from this pilot run, with some of the parameters, like the length of the burn-in phase and the thinning interval decided during the initial pilot run itself, with only the final sample size required deferred to the Raftery-Lewis analysis on this pilot run. Of course the parameter values settled upon during this pilot run was later cross-checked against Raftery-Lewis, but it must be cautioned that this approach should be reserved for experienced users only, and newcomers to MCMC should take the approach mentioned in the above paragraph. The issues involved in determining these parameters of the MCMC runs is discussed below in detail.

The first thing one needs to decide is the length of an initial burn-in period so that one starts storing the data only after this initial burn-in phase is over with the hope that the chain has now stabilized to its invariant distribution. This may be decided by looking at the trace plot obtained by pulling down the item samples.. from the inference menu of WinBUGS, or keep updating till WinBUGS is saying that it is in adapting mode, or run Raftery-Lewis on a pilot run (the recommended approach). In this example WinBUGS was in an adapting mode till the first 4000 iterations. The dynamic trace plot indicated convergence problem only till the first couple of thousands iterations. But to be on the safer side I decided on a burn-in phase of 4000.

The next issue is for how long a chain is to be run. Of course the answer is, the more the merrier, but an idea about the minimum required sample size may be obtained using the Raftery-Lewis diagnostics tool on a pilot run. This problem of required run length also gets compounded with the fact that most of the subsequent estimation methods (based on the MCMC sample) depend on observations being independent, and the consecutive values from the MCMC run are not. This leads to the next consideration called the thinning interval.

With a thinning interval of k, one keeps storing every k-th value (after the initial burn-in phase) discarding the in between $k-1$ generated values. The value of this k may be obtained by studying plot of the autocorrelation function on a pilot MCMC run. Take k to be that number where the autocorrelation has almost died down to 0. Raftery-Lewis also provides an estimate of both the length of the chain and the number of independent samples required,

and thus giving one an idea about the optimal thinning interval. For this example, based on the autocorrelation plots, a k value of 50 seemed like a reasonable choice. But as mentioned above, the final sample size was decided based on Raftery-Lewis on this pilot run, which as also used to cross-check and validate the currently ad-hoc decision of a burn-in phase of 4000 and thinning interval of 50

Finally it is advisable to run multiple chains starting with disperse initial values, because many diagnostic convergence tests, most notably the Gelman-Rubin, require outputs from multiple chains. A minimum of 2 to up to 5 parallel chains may be run for these convergence diagnostic checks. Other than the convergence diagnostics, multiple chains also have some other benefits. Note that if the initial burn-in phase is small, one can forgo the issue of the thinning interval and run a large number (say, as suggested by Raftery-Lewis) of parallel chains starting from a disperse set of initial values and picking only the final value after the burn-in phase from each chain. These will give an i.i.d. sample from the invariant distribution. However in practice for complicated models, with parameters having a high degree of correlation, one encounters what is called slowly mixing chains. In such situations it typically takes a long burn-in period to achieve stationarity. Thus most often than not this approach does not turn out to be economical and one usually adapts the thinning interval approach from one long chain which approximates the i.i.d. property of the generated MCMC sample. However a mixed strategy where one runs a moderate number of moderate sized chains is possibly the most pragmatic approach that one can take. Here for the final run we have taken this mixed approach which is described towards the end of §2.3, but before that all these considerations are first summarized in Table 1 below.

Table 1: Some Considerations for Running MCMC

Issue	Meaning	Guideline
Burn-in Phase	How many initial samples to discard?	1. Run Raftery-Lewis on a pilot chain with no discards. 2. Look at the dynamic trace plots and continue till the plots look stationary *i.e.* oscillating around some mean value without exhibiting any trend. 3. Wait till `WinBUGS` says `adapting`.
Thinning Interval	How frequently to sample from the final chain?	1. Equals the "dependence factor" of the Raftery-Lewis output run on the pilot chain. 2. Look at the autocorrelation plot till it comes down to 0. The lag where the autocorrelation is 0, is the ideal thinning interval.
Chain length	How large a sample should we draw?	Run Raftery-Lewis on a pilot chain.
Parallel Chains	How many chains to run?	1. 2 to 5 for running the Gelman-Rubin diagnostics. 2. Derive from the sample size recommendation of Raftery-Lewis on a pilot chain and the time it takes to run a given chain length.

In this example in the pilot run we first ran 5 parallel chains with randomly generated initial values for each chain. Since WinBUGS was in the adapting mode for the first 4000 iterations, we decided on a burn-in period of 4000 *i.e.* the initial 4000 values are simply discarded and we started storing the generated values only thereafter. Then each of the 5 chains were run for 10,000 more iterations. This was simply an ad-hoc choice and the final required number of samples was decided only after running a Raftery-Lewis on these pilot samples. For checking convergence, all the beta's thus generated were output using coda. Note that we kept a thinning interval of 1 in spite of a significant autocorrelation till up to the 50-th lag because it will be taken care of in the subsequent analysis in R. Thus the script file used to run this MCMC in WinBUGS was as follows:

```
display(log)
check(My_Files/Trilogy/Example/model.txt)
data(My_Files/Trilogy/Example/datmat.txt)
data(My_Files/Trilogy/Example/prior.txt)
compile(5)
gen.inits()
update(4000)
set(beta)
update(10000)
thin.updater(1)
coda(*,My_Files/Trilogy/Example/output)
```

The log file log.odc was then interactively generated mostly using the inference menu of WinBUGS. However the subsequent analysis using the MCMC output is better done outside WinBUGS using R, which we take up next.

2.3 Convergence Diagnostics

As mentioned in §2.2, it is better to use WinBUGS only for generating the MCMC samples. WinBUGS, though tries to provide some of the subsequent analyses, features like tools for convergence diagnostics and inference about not just the parameters themselves but other parametric functions of interest like the π_{ij}'s of this example, are either very weak or non-existent in WinBUGS. For these analyses it is best to request WinBUGS to output the MCMC samples as plain text files using its coda command (as in the last statement of the above script file or by pulling down samples.. from its inference menu) and then read these files into R using one of its coda suit of functions. Thus we now first get into R, upload its coda library and start the next stage of analysis in there as follows.

```
> library(coda)
> d1<-read.coda("output1.txt","outputIndex.txt",thin=50)
> d2<-read.coda("output2.txt","outputIndex.txt",thin=50)
> d3<-read.coda("output3.txt","outputIndex.txt",thin=50)
> d4<-read.coda("output4.txt","outputIndex.txt",thin=50)
> d5<-read.coda("output5.txt","outputIndex.txt",thin=50)
```

I prefer to work from scratch (because it gives me more control over the things I want to do), but if you want, you can also issue the R command codamenu() (of course only after attaching the coda library to your current R workspace) and then work interactively with it to read the files, run convergence diagnostics and do only those things this menu allows you to do. Each of the 5 output files contains 10,000 MCMC samples generated from the joint posterior of the $2 \times 5 = 10$ beta's for the 5 chains we have run (specified as the compile(5) command in the script file). But these output files are not organized in a spreadsheet format with 10,000 rows and 10 columns, with each column corresponding to a beta. For each chain WinBUGS outputs all the 10,000×10=100,000 values in one single column of a file (and thus as many files as the number of chains run) and then supplies an index file (outputIndex.txt in our case) stating which sets of rows correspond to which beta's, which is required and utilized by read.coda. codamenu() will also prompt you asking for the name of this index file, when you ask it to read an MCMC output file generated by WinBUGS.

Now recall from the autocorrelation plots in log.odc that we should have possibly used a thinning interval of 50, whereas while running the WinBUGS script we had kept the thinning interval as 1 (using the thin.updater(1) command mentioned in the script file). It was also mentioned there in §2.2 that it will be rectified later in R. This is precisely what the option thin=50 is doing in the read.coda commands above. That is in the end, for j=1,..,5, each of the dj's will contain a 200×10 matrix with the columns corresponding to the beta values and the rows corresponding to the iteration numbers. Since originally there were 10,000 values for each beta, and we are using a thinning interval of 50, it will contain 200 rows, with the first row corresponding to iteration number 4001, second row corresponding to iteration number 4051 etc. to the last or 200-th row corresponding to iteration number 13951. It starts at 4001 because we had used a burn-in phase of 4000.

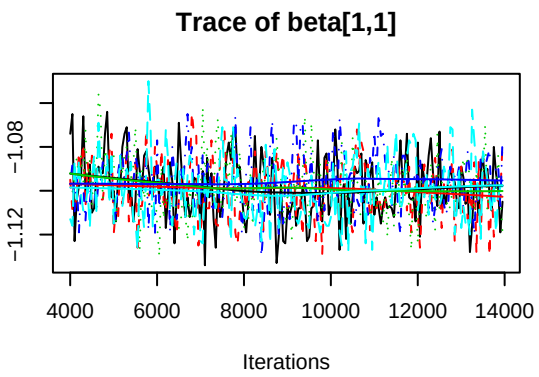
Now we are ready to proceed to the next phase. But before doing so we shall first collate the 5 dj's into a single MCMC object of R using the command

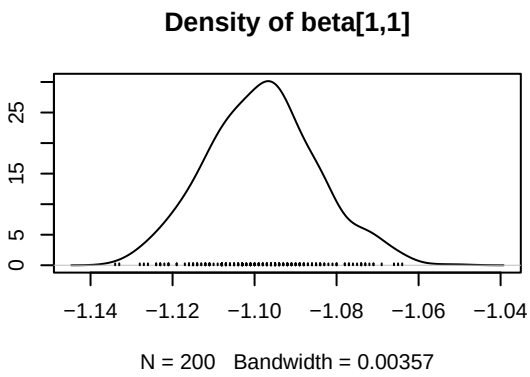
```
> d<-mcmc.list(d1,d2,d3,d4,d5)
```

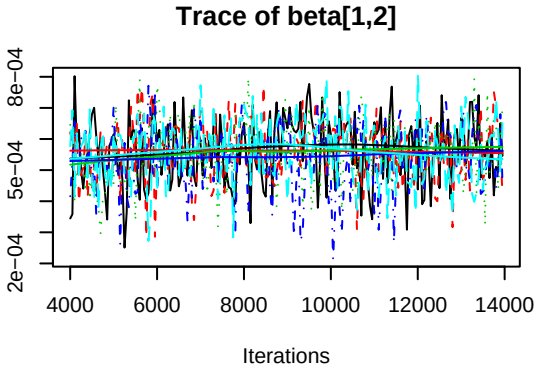
so that the diagnostic checks requiring multiple chain outputs like Gelman-Rubin find them in one place. Actually Gelman-Rubin is the only formal convergence diagnostic test that WinBUGS also provides, but we shall nonetheless demonstrate all the features that the coda library of R provides for the MCMC outputs from WinBUGS, with the risk of some repetitions. We start with some plots with the commands

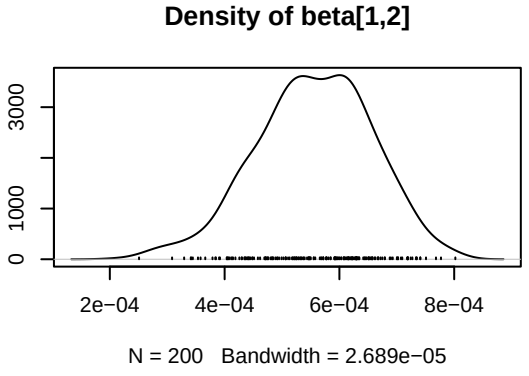
```
> plot(mcmc.list(d1[,1:4],d2[,1:4],d3[,1:4],d4[,1:4],d5[,1:4]))
> plot(mcmc.list(d1[,5:8],d2[,5:8],d3[,5:8],d4[,5:8],d5[,5:8]))
> plot(mcmc.list(d1[,9:10],d2[,9:10],d3[,9:10],d4[,9:10],d5[,9:10]))
```

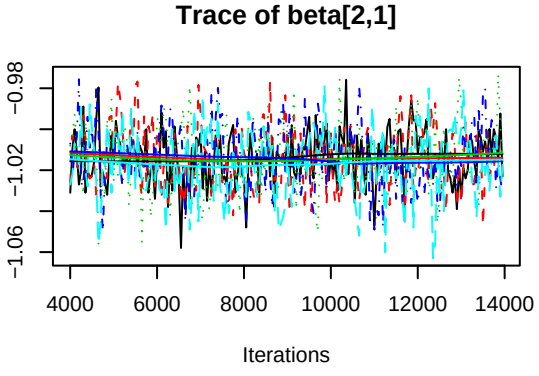
These three commands were actually issued for embedding the plots in this document for creating three different postscript files. In an interactive session this is not required and a simple s plot(d,ask=T) will produce these 20 plots screen by screen.

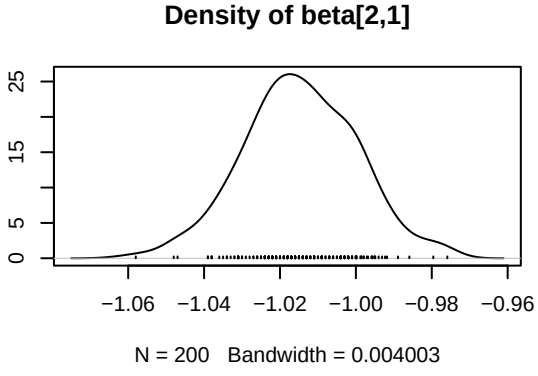

Trace of beta[1,1]
−1.08
−1.12
4000
6000
8000
10000
12000
14000
Iterations

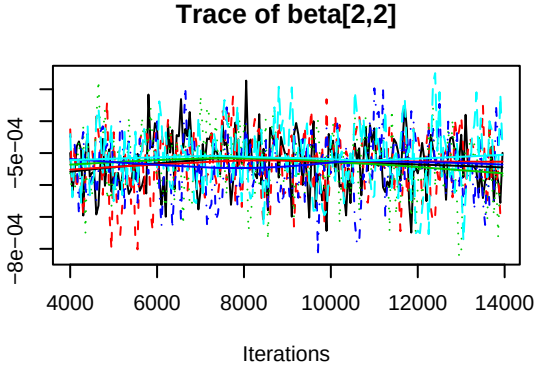

Density of beta[1,1]
0
5
15
25
−1.14
−1.12
−1.10
−1.08
−1.06
−1.04
N = 200 Bandwidth = 0.00357

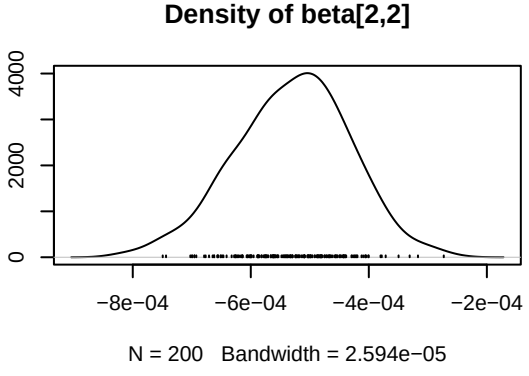

Trace of beta[1,2]
2e−04
5e−04
8e−04
4000
6000
8000
10000
12000
14000
Iterations

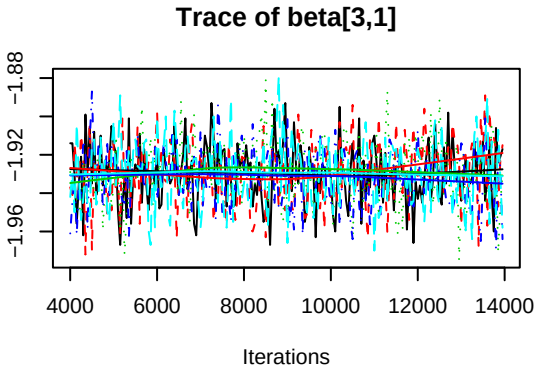

Density of beta[1,2]
0
1000
3000
2e−04
4e−04
6e−04
8e−04
N = 200 Bandwidth = 2.689e−05

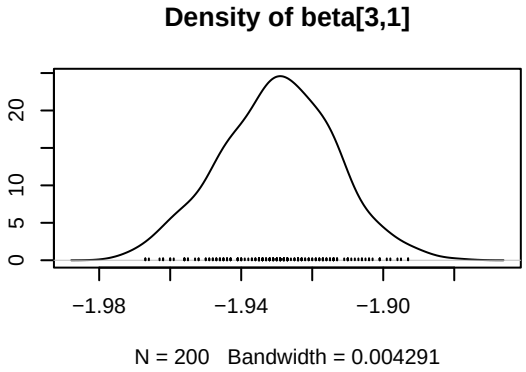

Trace of beta[2,1]
−1.06
−1.02
−0.98
4000
6000
8000
10000
12000
14000
Iterations

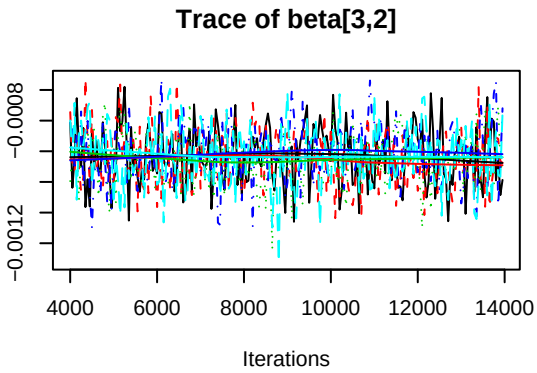

Density of beta[2,1]
0
5
15
25
−1.06
−1.04
−1.02
−1.00
−0.98
−0.96
N = 200 Bandwidth = 0.004003

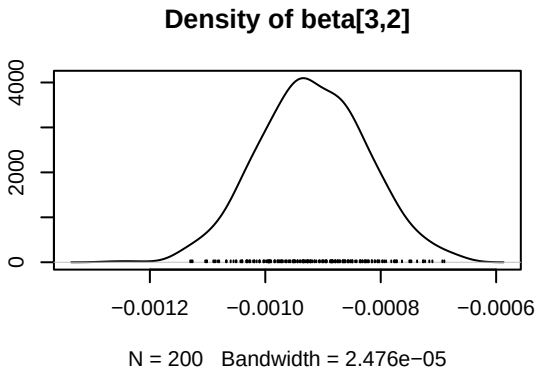

Trace of beta[2,2]
−8e−04
−5e−04
4000
6000
8000
10000
12000
14000
Iterations

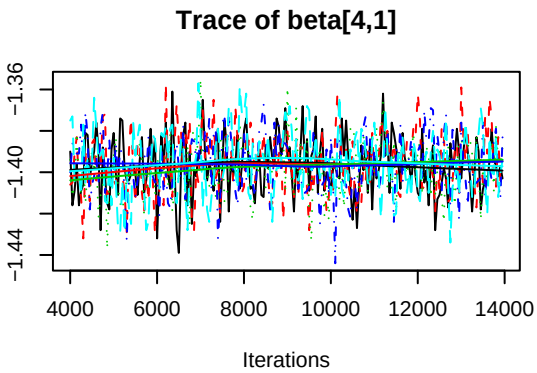

Density of beta[2,2]
0
2000
4000
−8e−04
−6e−04
−4e−04
−2e−04
N = 200 Bandwidth = 2.594e−05

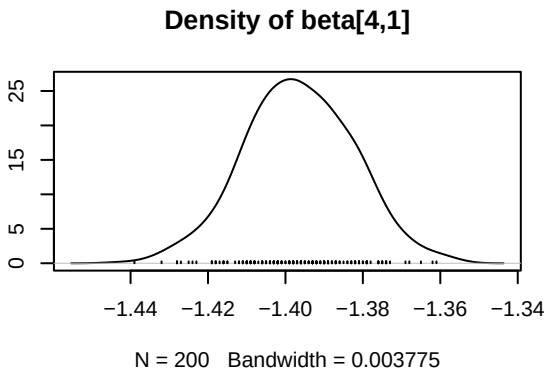

Trace of beta[3,1]
−1.88
−1.92
−1.96
4000
6000
8000
10000
12000
14000
Iterations

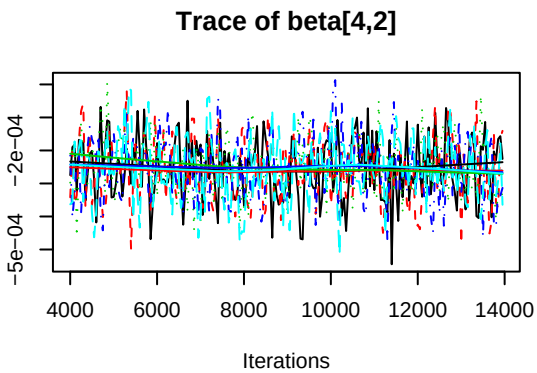

Density of beta[3,1]
0
5
10
20
−1.98
−1.94
−1.90
N = 200 Bandwidth = 0.004291

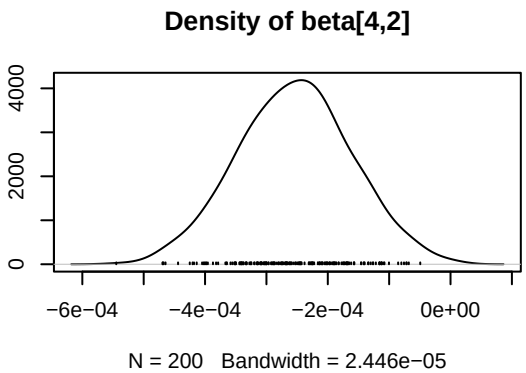

Trace of beta[3,2]
−0.0008
−0.0012
4000
6000
8000
10000
12000
14000
Iterations

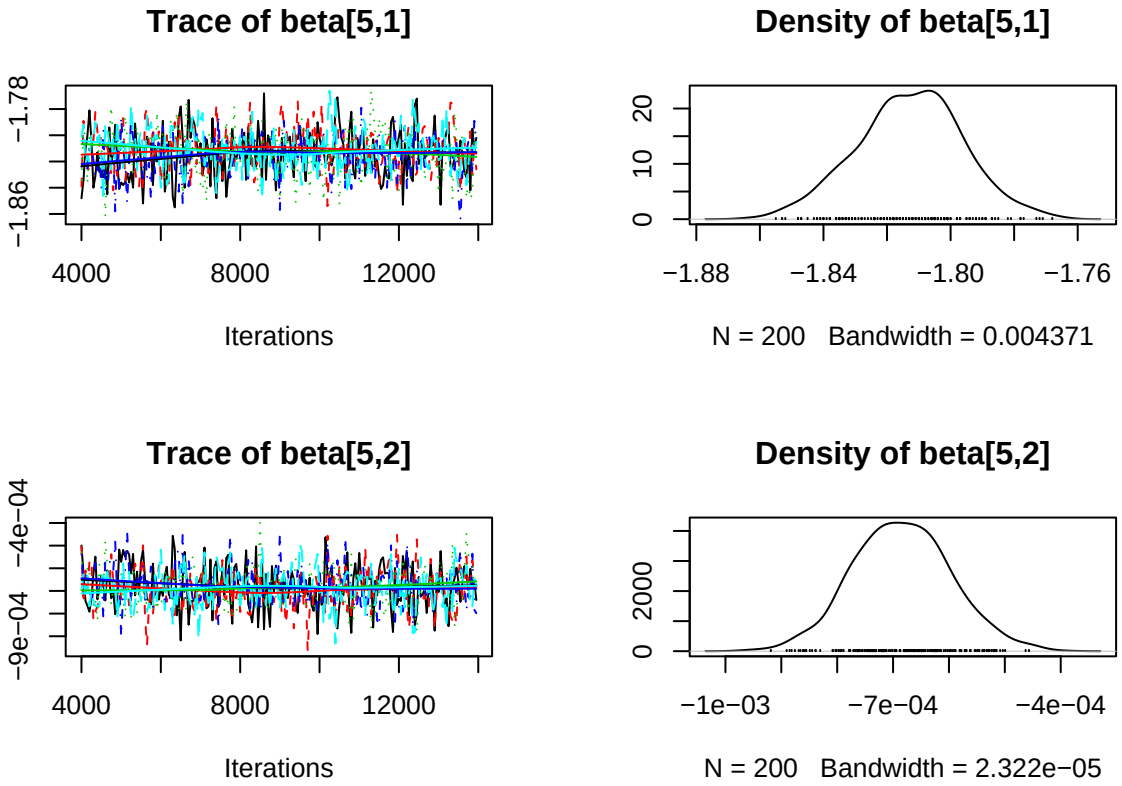

Density of beta[3,2]
0
2000
4000
−0.0012
−0.0010
−0.0008
−0.0006
N = 200 Bandwidth = 2.476e−05


Trace of beta[4,1]
−1.36
−1.40
−1.44
4000
6000
8000
10000
12000
14000
Iterations


Density of beta[4,1]
0
5
15
25
−1.44
−1.42
−1.40
−1.38
−1.36
−1.34
N = 200 Bandwidth = 0.003775


Trace of beta[4,2]
−2e−04
−5e−04
4000
6000
8000
10000
12000
14000
Iterations


Density of beta[4,2]
0
2000
4000
−6e−04
−4e−04
−2e−04
0e+00
N = 200 Bandwidth = 2.446e−05



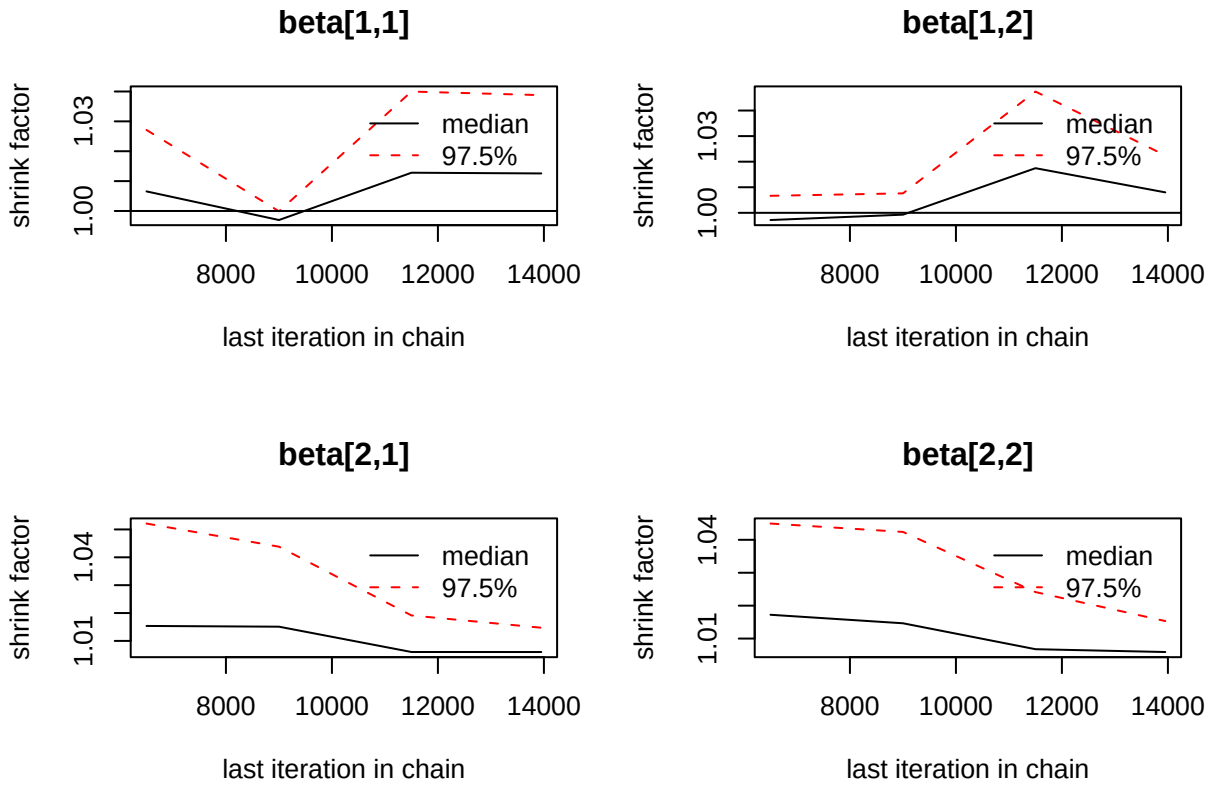
The `plot` command for an `mcmc` object by default produces these trace plots and density estimates. These plots are also available in `WinBUGS` from the `samples..` item of its `inference` menu. The thing to note is that from these trace plots (or the `history` plot in `WinBUGS`) it appears that the chain has converged to its stationary distribution. This is the first rough check for convergence and one of the most useful ones. Next we start with some formal diagnostic checks.

The first one of such is the Gelman-Rubin scale reduction factor. The idea behind this diagnostic check is as follows. Suppose one runs m parallel chains each of length n for a certain parameter θ. Now consider the quantity σ^2, the posterior variance of θ. Two different estimates of σ^2 may be constructed. The first estimate, say s_1^2 is the variance of all the mn values pooled together, and the second one, say s_2^2 is the average variance of the m chains, each one computed using the n values in the chain. If all the chains have converged to the stationary distribution then s_1^2 and s_2^2 will be close to each other yielding a common estimate of σ^2. Otherwise, s_1^2 will overestimate and s_2^2 will underestimate σ^2. This is because in that case the values sampled by the m chains will be far apart from each other depending on the initial values with which each chain was started, and thus as a result of pooling all the mn values together s_1^2 will yield an overestimate of σ^2. On the other hand in this case, where the chains have not converged, the m chains have not had time to traverse the entire posterior spectrum of θ and as a result the variance of each individual chain and thus their average s_2^2 will underestimate σ^2. Gelman-Rubin's scale reduction factor is the ratio $r \approx s_1^2/s_2^2$[2].

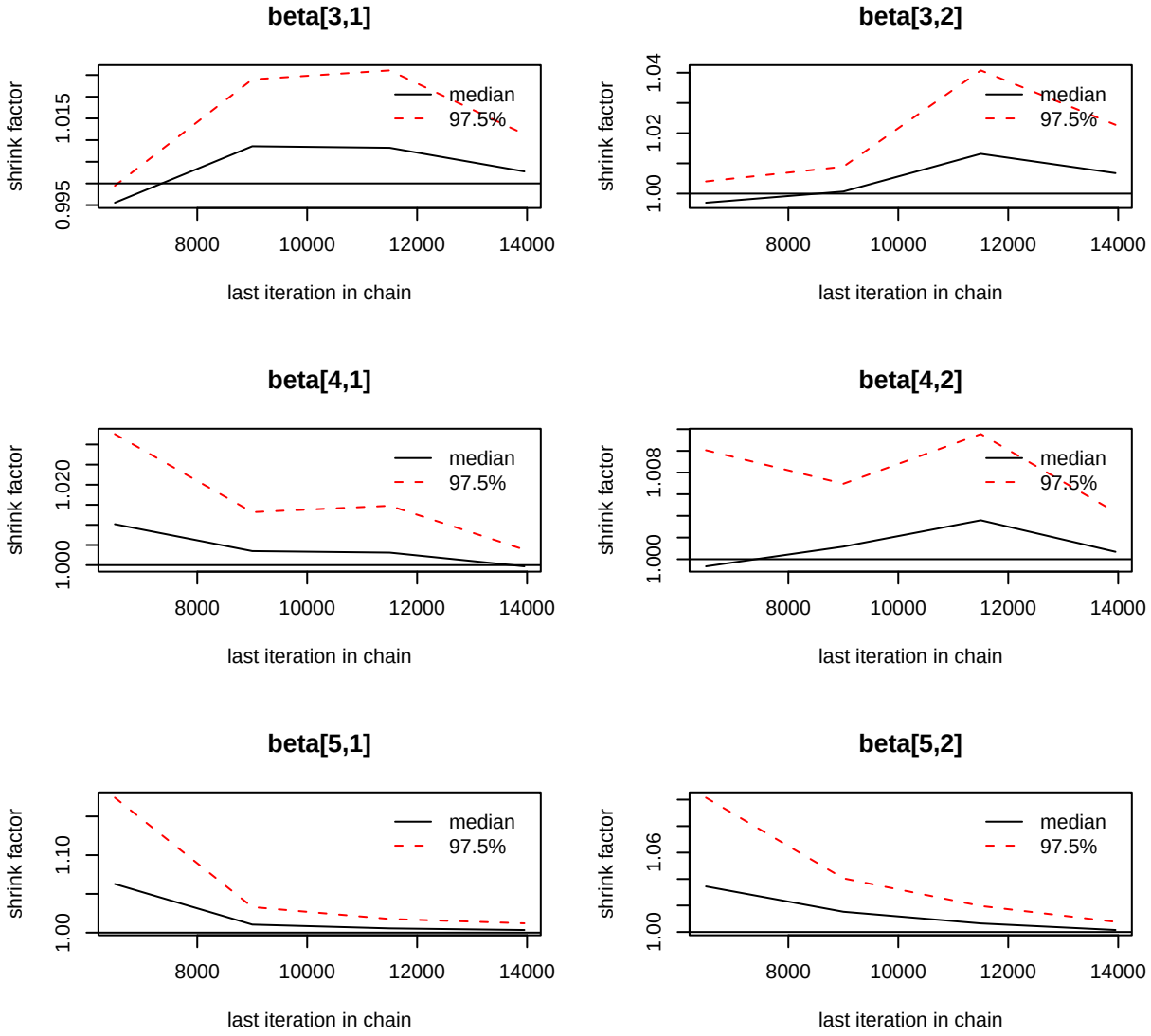
[2]Actually the exact formula is a little more complicated, which is based on Analysis of Variance results and approximation of the posterior of θ using a t distribution. Whatever the exact formula might be, the

Thus in the beginning one would expect to see an r value much larger than 1. Then as the iteration proceeds the r value should approach 1. A point estimate and a 97.5% quantile (an upper bound) of r at the end of the iteration and their evolution as the iteration proceeds is produced by the following commands.

```
> gelman.diag(d)
Potential scale reduction factors:

          Point est. 97.5% quantile
beta[1,1]       1.01           1.04
beta[1,2]       1.01           1.02
beta[2,1]       1.01           1.01
beta[2,2]       1.01           1.02
beta[3,1]       1.00           1.01
beta[3,2]       1.01           1.02
beta[4,1]       1.00           1.00
beta[4,2]       1.00           1.00
beta[5,1]       1.00           1.01
beta[5,2]       1.00           1.01
> gelman.plot(mcmc.list(d1[,1:4],d2[,1:4],d3[,1:4],d4[,1:4],d5[,1:4]))
> gelman.plot(mcmc.list(d1[,5:10],d2[,5:10],d3[,5:10],d4[,5:10],d5[,5:10]))
```



idea behind the Gelman-Rubin convergence diagnostic check is as described above.



Again just one command `gelman.plot(d)` would do for the interactive session. To produce hard-copies or embedding them in documents you need to suitably break-up the plots. The r values look fairly satisfactory once more confirming convergence. These plots are also produced by `WinBUGS` (use `bgr diag` after pulling down the `inference` menu's `samples...`) and thus so far we have not got anything additional in the `coda` suit of tools.

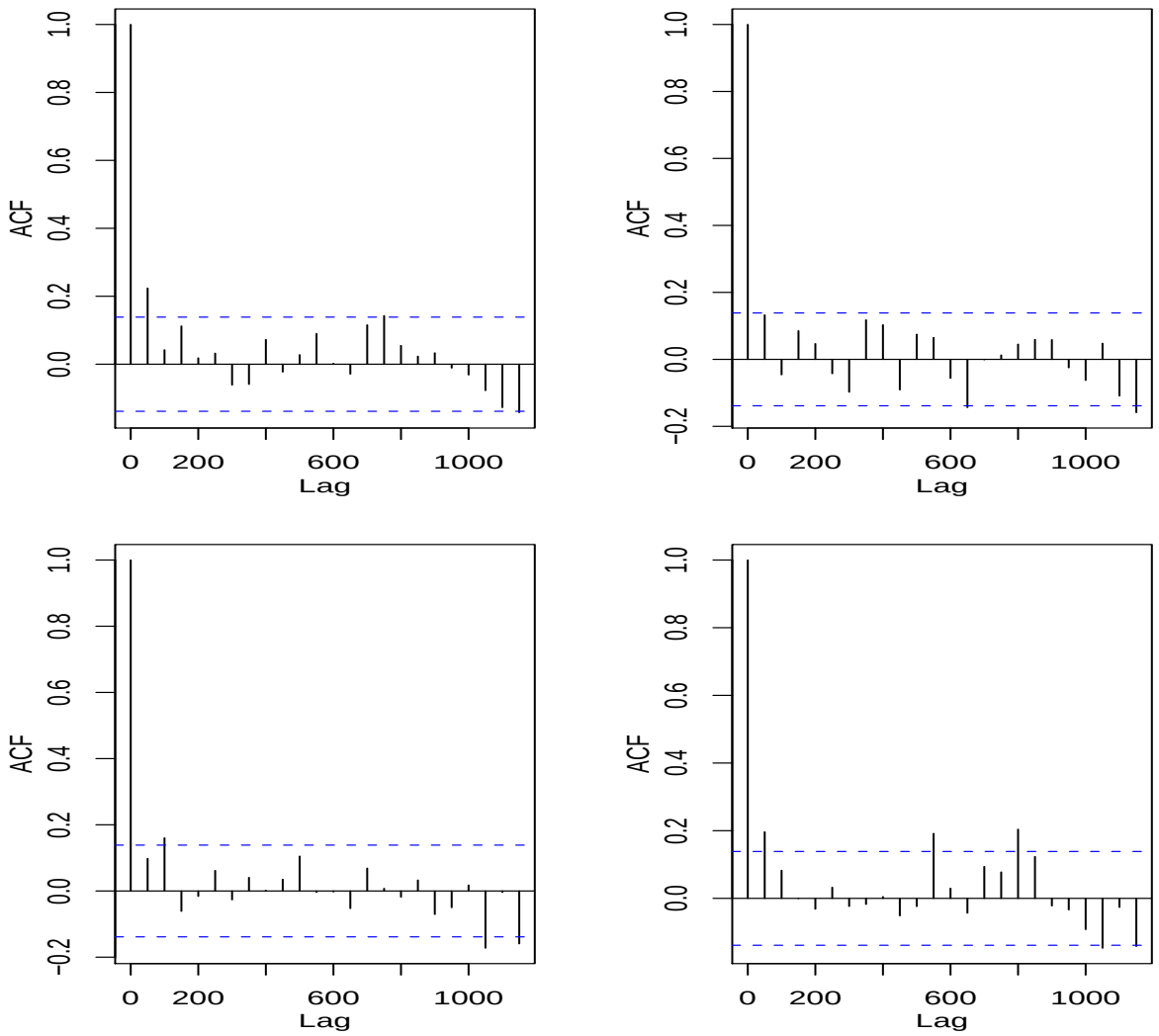
Now we shall look at a few more convergence diagnostic tools which are only available in the `coda` library of `R`, which are not available in `WinBUGS`. Also since these diagnostics do not require multiple chains we shall only apply them on one of the chains, to reduce the volume of the output. Results of these diagnostics checks obtained for other chains are similar.

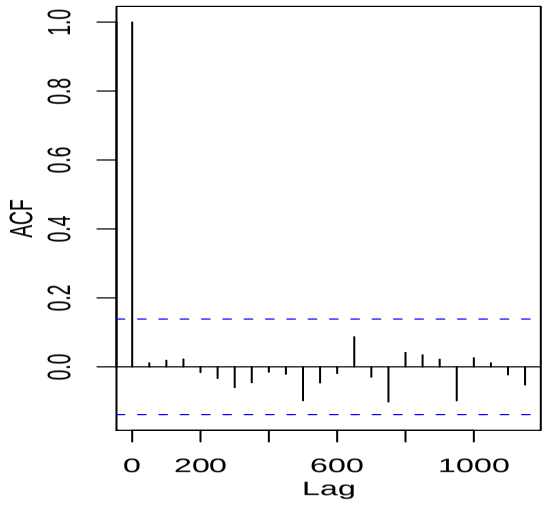
The first such check is due to Geweke. The idea behind this check is as follows. The first and the last part of a chain is obtained. By default one takes the first 10% and the last 50% of the chain. Then one examines whether the means of these two parts are same or not using a Z-statistic. Since the values are autocorrelated the usual Z-test does not work. Geweke's suggestion was to estimate the variance using the spectral density at 0 for the

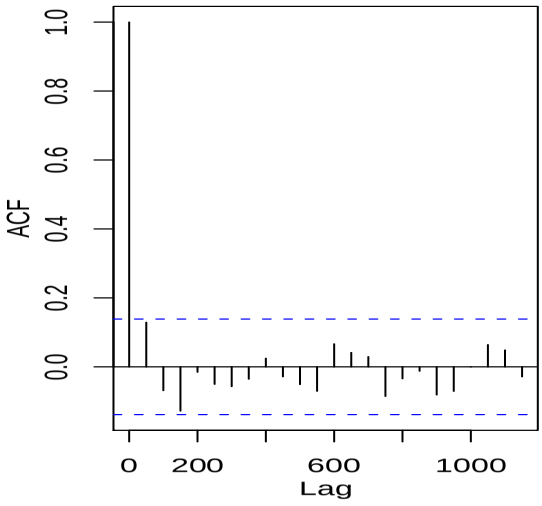
estimated standard error in the difference of the two means. The diagnostic test essentially churns out a Z-statistic. If the value of this Z-statistic is moderate, say within ± 2, then one can conclude that there is no significant difference in the mean values of the two tails of the chain and thus fulfilling another criterion for convergence. This is done as follows.

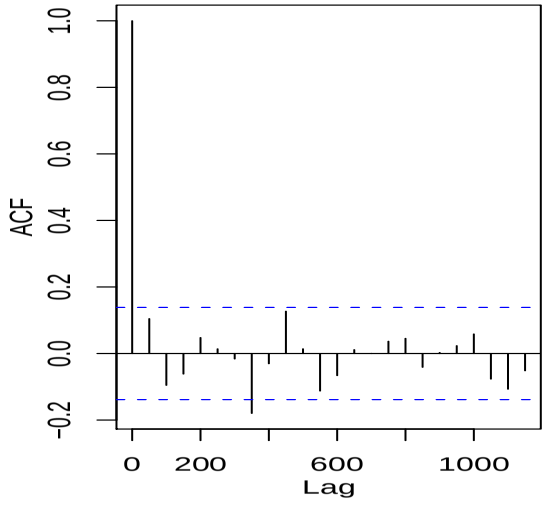
```
> geweke.diag(d1)
Fraction in 1st window = 0.1
Fraction in 2nd window = 0.5
beta[1,1] beta[1,2] beta[2,1] beta[2,2] beta[3,1] beta[3,2] beta[4,1] beta[4,2]
   1.4727   -0.7480   -0.7102   -0.4309    1.5776   -0.6249   -0.5872    1.0667
beta[5,1] beta[5,2]
  -1.1927    0.2419
```

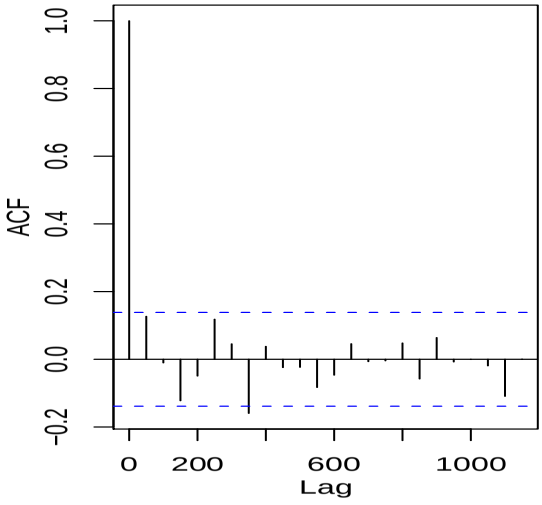
Since all the Z-values appear to be quite nice, the chain seems to have converged, according to the Geweke criterion as well. Since Geweke criterion brings up the issue of autocorrelation, at this point it is worth examining the autocorrelation of the `d1` chain. The j-th row of the following plots gives the autocorrelation plots of `beta[j,1]` and `beta[j,2]`.

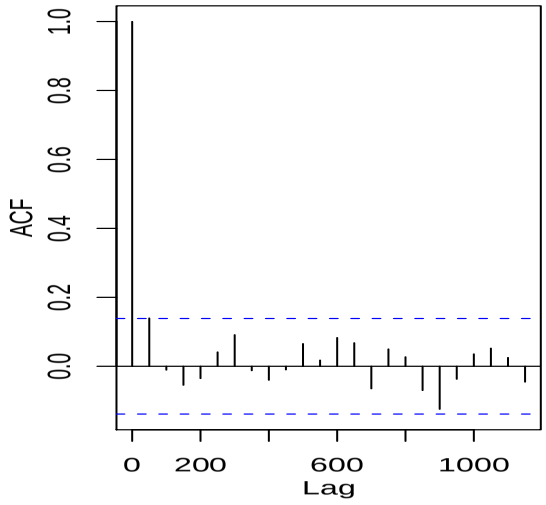


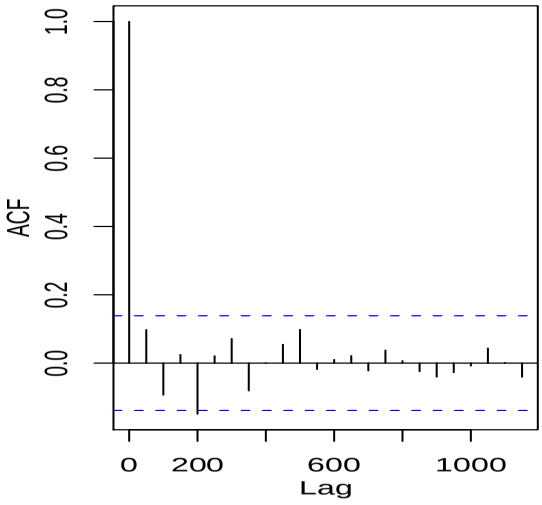

ACF
1.0
0.8
0.6
0.4
0.2
0.0
0
200
600
1000
Lag


ACF
1.0
0.8
0.6
0.4
0.2
0.0
0
200
600
1000
Lag


ACF
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
200
600
1000
Lag


ACF
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
200
600
1000
Lag


ACF
1.0
0.8
0.6
0.4
0.2
0.0
0
200
600
1000
Lag


ACF
1.0
0.8
0.6
0.4
0.2
0.0
0
200
600
1000
Lag

Note how these autocorrelation plots of the thinned chain are different from the ones provided in `log.odc` for the unthinned chain. Here the values have become almost i.i.d's.

The next diagnostic test is given by Heidelberger and Welch. This convergence test uses the Cramer-von-Mises statistic to test the null hypothesis that the sampled values come from a stationary distribution. The test is successively applied, first to the whole chain, then after discarding the first 10%, 20%, ... of the chain until either the null hypothesis is accepted, or 50% of the chain has been discarded. The latter outcome constitutes "failure" of the stationarity test and indicates that a longer MCMC run is needed. The half-width test calculates a 95% confidence interval for the mean, using the portion of the chain which passed the stationarity test. Half the width of this interval is compared with the estimate of the mean. If the ratio between the half-width and the mean is lower than 0.1, the half-width test is passed. Otherwise the confidence interval is deemed to be too wide so that the length of the sample is considered to be not long enough for estimation of the mean with sufficient accuracy.

```
> heidel.diag(d3)

          Stationarity start     p-value
          test         iteration
beta[1,1] passed       21        0.3490
beta[1,2] passed       21        0.5260
beta[2,1] passed        1        0.3786
beta[2,2] passed        1        0.0734
beta[3,1] passed        1        0.3902
beta[3,2] passed        1        0.5573
beta[4,1] passed       21        0.2023
beta[4,2] passed        1        0.1113
beta[5,1] passed        1        0.3952
beta[5,2] passed        1        0.3222

          Halfwidth Mean      Halfwidth
          test
beta[1,1] passed    -1.099033 2.00e-03
beta[1,2] passed     0.000566 1.18e-05
beta[2,1] passed    -1.014321 2.63e-03
beta[2,2] passed    -0.000533 1.60e-05
beta[3,1] passed    -1.928755 2.39e-03
beta[3,2] passed    -0.000928 1.29e-05
beta[4,1] passed    -1.397056 2.12e-03
beta[4,2] passed    -0.000248 1.19e-05
beta[5,1] passed    -1.812545 2.99e-03
beta[5,2] passed    -0.000684 1.45e-05
```

Thus our chain passed this test as well. Thus now having been convinced about convergence we are finally in a position to request for a Raftery-Lewis recommendations for the final sample size and a cross-check on the already decided burn-in phase of 4000 and a thinning

interval of 50. But recall from the discussion in the second paragraph in page 11, that this pilot run should contain consecutive values from the chain with a thinning interval of 1. This is required by Raftery-Lewis in order to get an assessment of the autocorrelation structure of the chain which is required for providing recommendations for the thinning intervals. Thus we cannot use the dj's for this purpose, since they already have a thinning interval of 50. But all we have to do for this is re-read one of the original `outputj.txt` files produced by the `coda` command of `WinBUGS` (and this was precisely the reason for setting the `thin.updater` as 1 in the `script` file in page 13), and then run a Raftery-Lewis on this as follows.

```
> pilot<-read.coda("output1.txt","outputIndex.txt")
> raftery.diag(pilot)

Quantile (q) = 0.025
Accuracy (r) = +/- 0.005
Probability (s) = 0.95

           Burn-in  Total Lower bound  Dependence
           (M)      (N)   (Nmin)       factor (I)
 beta[1,1] 40       48985 3746         13.10
 beta[1,2] 31       33192 3746          8.86
 beta[2,1] 38       45442 3746         12.10
 beta[2,2] 35       37791 3746         10.10
 beta[3,1] 29       32997 3746          8.81
 beta[3,2] 29       31504 3746          8.41
 beta[4,1] 32       36977 3746          9.87
 beta[4,2] 29       30738 3746          8.21
 beta[5,1] 47       52471 3746         14.00
 beta[5,2] 34       37781 3746         10.10
```

Above Raftery-Lewis output gives `M`, the size of the burn-in phase, `N`, the total length of the chain, `Nmin`, the number of *independent* samples, and `I`, the dependence factor = `(M+N)/Nmin` equaling the thinning interval length; that is required, in order to estimate the 0.025-th quantile of the respective posteriors with an error of ± 0.005 with 95% confidence. Note that our burn-in phase of 4000 and thinning interval of 50 is well beyond the Raftery-Lewis recommendations which are in the order of 40 and 15 respectively, and thus we can indeed take the burnt-in, thinned values in the 5 dj's as genuine i.i.d. samples from the posteriors of the respective `beta`'s.

But now note that each dj contains just 200 observations on the respective `beta`'s and even after pooling them together we only have a sample of size 1000 on each `beta`'s, whereas Raftery-Lewis recommends a minimum independent sample of size 3746. Thus we need to generate more sample. Since we already have an independent sample of size 1000 (from the 5 dj's) we need to generate about 3000 more such sample. Thus following the Raftery-Lewis recommendation now one can either run a chain of length 45,040 and then take every 15-th observation after discarding the first 40, or run an equivalent number of parallel chains. Here however, we decided to stick to our original burn-in phase of 4000, thinning interval of 50 and chains of post burn-in length 10,000. The main reason for doing this is, we know that we

can achieve convergence and approximate i.i.d.-ness with these parameter values. Now if we start with some new parameters following the Raftery-Lewis recommendation, then though the computational effort will substantially reduce, we have to again check for convergence in this new set-up which is not guaranteed. Since one run with our parameters yield only 200 independent samples, we decided to run 15 parallel chains with these parameters to garner 3000 additional samples from the posteriors of the `beta`'s. Thus we modified our original `script` file as follows

```
display(log)
check(My_Files/Trilogy/Example/model.txt)
data(My_Files/Trilogy/Example/datmat.txt)
data(My_Files/Trilogy/Example/prior.txt)
compile(15)
gen.inits()
update(4000)
set(beta)
update(10000)
thin.updater(1)
coda(*,My_Files/Trilogy/Example/output2)
```

and ran this script in `WinBUGS` to generate 3000 additional sample on each `beta`'s, which are then read into `R` using the commands

```
> d6<-read.coda("output21.txt","outputIndex.txt",thin=50)
> d7<-read.coda("output22.txt","outputIndex.txt",thin=50)
.....................................................
.....................................................
> d19<-read.coda("output214.txt","outputIndex.txt",thin=50)
> d20<-read.coda("output215.txt","outputIndex.txt",thin=50)
```

2.4 Posterior Inference

Now with the samples of required size in place and convergence in order (it is not necessary but nonetheless safe to recheck convergence using trace plots, `gelman.diag`, `geweke.diag` and `heidel.diag` on these new `d6` to `d20`, which I did and was satisfied), we are finally ready to draw inference about the π_{ij}'s, the main objective of the study. The first step towards that is to collate the `beta` values scattered across the 20 `dj`'s in one place, which is done as follows in `R`.

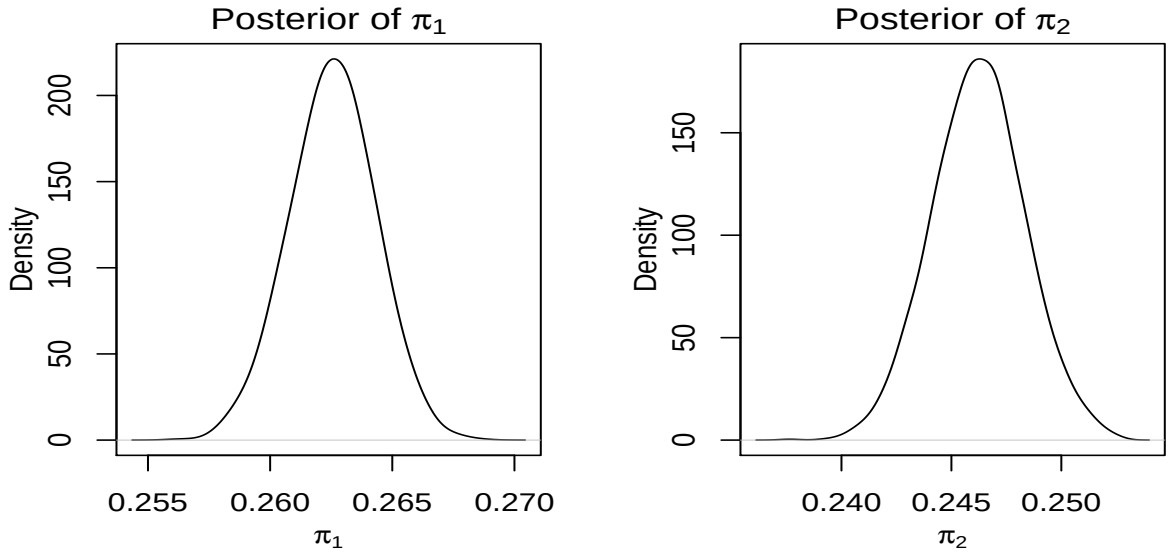
```
> beta11<-c(d1[,"beta[1,1]"],d2[,"beta[1,1]"],d3[,"beta[1,1]"],d4[,"beta[1,1]"],
       d5[,"beta[1,1]"],d6[,"beta[1,1]"],d7[,"beta[1,1]"],d8[,"beta[1,1]"],
       d9[,"beta[1,1]"],d10[,"beta[1,1]"],d11[,"beta[1,1]"],d12[,"beta[1,1]"],
       d13[,"beta[1,1]"],d14[,"beta[1,1]"],d15[,"beta[1,1]"],d16[,"beta[1,1]"],
       d17[,"beta[1,1]"],d18[,"beta[1,1]"],d19[,"beta[1,1]"],d20[,"beta[1,1]"])
.....................................................................
.....................................................................
> beta52<-c(d1[,"beta[5,2]"],d2[,"beta[5,2]"],d3[,"beta[5,2]"],d4[,"beta[5,2]"],
       d5[,"beta[5,2]"],d6[,"beta[5,2]"],d7[,"beta[5,2]"],d8[,"beta[5,2]"],
```

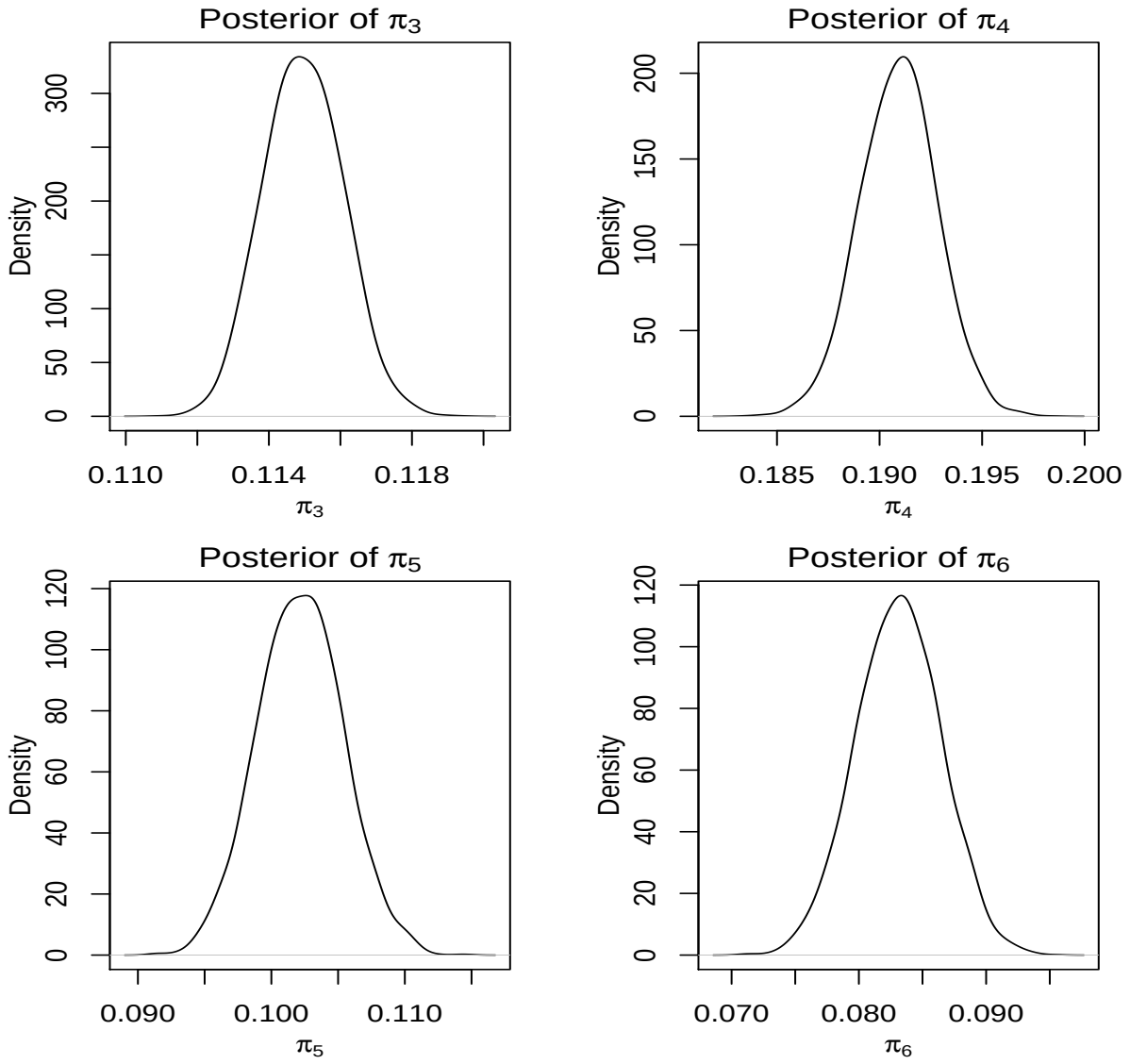
```
d9[,"beta[5,2]"],d10[,"beta[5,2]"],d11[,"beta[5,2]"],d12[,"beta[5,2]"],
d13[,"beta[5,2]"],d14[,"beta[5,2]"],d15[,"beta[5,2]"],d16[,"beta[5,2]"],
d17[,"beta[5,2]"],d18[,"beta[5,2]"],d19[,"beta[5,2]"],d20[,"beta[5,2]"])
```

Now consider a certain DMA on a certain month with (117,196,121,186,528,179) as its 6 x-values for the 6 trims. These were the actual values for DMA 1 in month 3. Given this $\boldsymbol{x}$ and the posterior of the `beta`'s we wish to draw posterior inference about the "true demand" π_j of the 6 trims for $j = 1, \ldots, 6$. Given $\boldsymbol{x}$, π_j's can be determined using (1) and evaluating this for each sampled value of the `beta`'s will give us the corresponding posterior of π_j's. These computations can be implemented in `R` as follows.

```
> beta<-array(c(beta11,beta12,beta21,beta22,beta31,beta32,beta41,beta42,
                beta51,beta52),dim=c(4000,2,5))
> x<-c(117,196,121,186,528,179)
> pi<-matrix(nrow=4000,ncol=6)
> for(j in 1:5)
+ {
+  for(i in 1:4000)
+  {
+   pi[i,j]<-1/(1+exp(-beta[i,1,j]-beta[i,2,j]*x[j]))
+  }
+ }
> for(i in 1:4000)
+ {
+  pi[i,6]<-1-sum(pi[i,1:5])
+ }
> plot(density(pi[,1]))
.......................
.......................
> plot(density(pi[,6]))
```

gives the following posterior densities of the π_j's.






```
> colStats(pi,mean)
[1] 0.26255865 0.24622957 0.11496725 0.19092746 0.10218983 0.07593548
> colStats(pi,median)
[1] 0.26257294 0.24621709 0.11494511 0.19096778 0.10216789 0.08314826
> colStats(pi,sd)
[1] 0.001752156 0.002119291 0.001105897 0.001866564 0.003191142 0.003333501
> HPDinterval(mcmc(matrix(c(1:4000,pi[,1]),ncol=2)))
         lower        upper
     0.2591790    0.2660495
> HPDinterval(mcmc(matrix(c(1:4000,pi[,2]),ncol=2)))
         lower        upper
     0.2421242    0.2504512
> HPDinterval(mcmc(matrix(c(1:4000,pi[,3]),ncol=2)))
         lower        upper
```

```
     0.1127959    0.1170288
> HPDinterval(mcmc(matrix(c(1:4000,pi[,4]),ncol=2)))
         lower        upper
     0.1875233    0.1948857
> HPDinterval(mcmc(matrix(c(1:4000,pi[,5]),ncol=2)))
          lower        upper
     0.09584511   0.1082404
> HPDinterval(mcmc(matrix(c(1:4000,pi[,6]),ncol=2)))
          lower        upper
     0.07637934   0.088926214
```

Recalling that the original trim 4 is $j = 6$ and trims 5 and 6 are $j = 4$ and 5 respectively it may be stated that the demands of trims 1 and 2 are highest and comparable, followed by trims 5, 3 and 6 with the last two being comparable, while the demand for trim 4 is the least. Interestingly the ADI values for the 6 trims for this DMA was (0.2349, 0.1635, 0.1919, 0.0929, 0.2046, 0.1122) none of which are contained in the 95% HPD interval. The trim-wise sales value for this observation was (1, 1, 3, 3, 0, 0, 3).