4. PROBIT MODEL FOR MULTINOMIAL DATA

Now we discuss the ordinal multinomial ordinal model for probit regression. This model is an extension of the binomial case, used when the response can take $J > 2$ values. Again, suppose we have N independent random variables y_i, but in the multinomial case they can take values in $0, 1, 2$.

We call:

$$p_{ij} = P(Y_i = j)$$

which is the probability that the i-th obesrvation falls into category j, and

$$\theta_{ij} = P(Y_i \leq j)$$

which is the cumulative probability.

As in the binary case we assume the link function is the cdf of a normal distribution

$$p_i = g^{-1}(\gamma_j - \beta^T \mathbf{x}_i)$$

Where γ_j is the intercept of the j-th level, if x_i doesn't include an intercept. The **probit model** is obtained if we define $g^{-1}(.) = \mathbf{\Phi}(.)$, where $\mathbf{\Phi}(.)$ denotes the **cumulative distribution function of the standard normal distribution**, so we get:

$$p_i = g^{-1}(\gamma_j - \beta^T \mathbf{x}_i) = \Phi(\gamma_j - \beta^T \mathbf{x}_i) \leftrightarrow \Phi^{-1}(p_i) = \gamma_j - \beta^T \mathbf{x}_i \qquad\qquad j = 1, ..., J-1$$

4.1 AUGMENTED MODEL

The most natural way to view ordinal data is to postulate the existence of a latent variable associated with each response. The latent varibale interpretation allows posterior inference via Gibbs Sampling. For each $Y_i \in \{0, 1\}$ we assume a prior distribution for Z:

$$Z_i|\beta \overset{ind}{\sim} N(\beta^T \mathbf{x}_i, 1) \qquad\qquad i = 1, ..., n$$

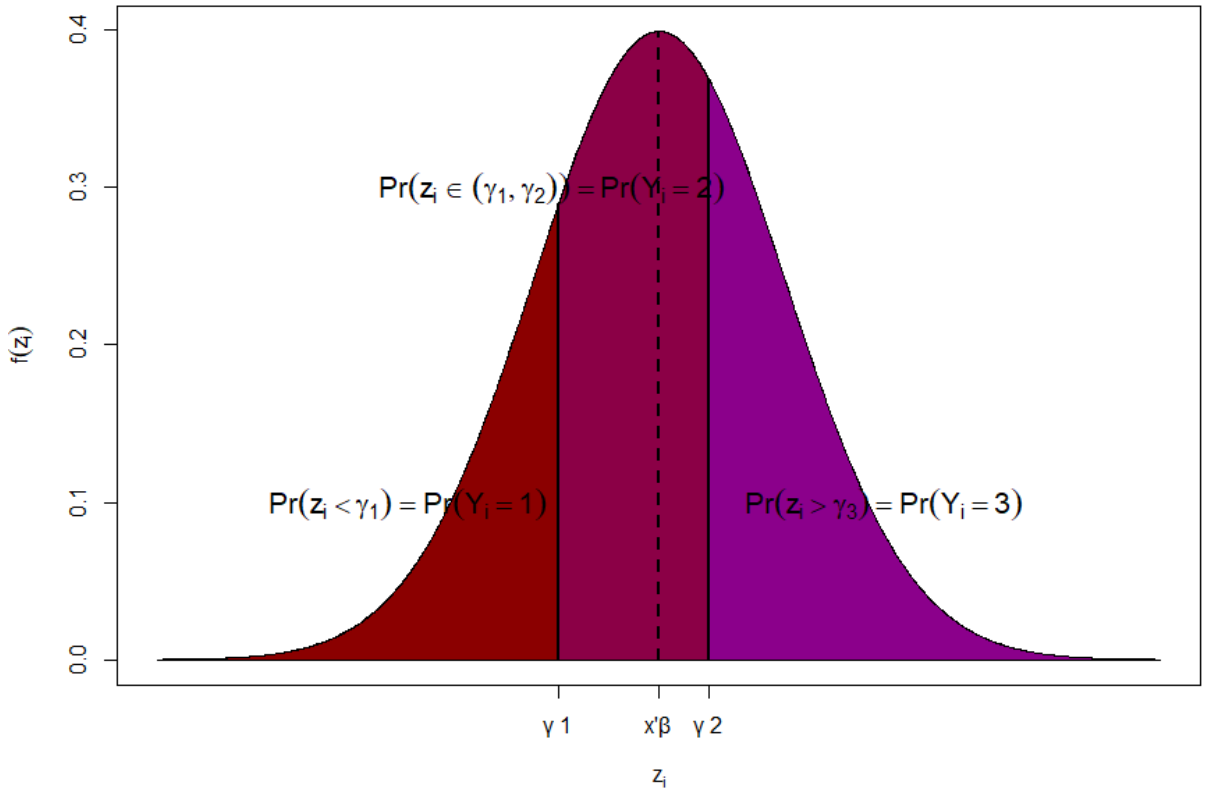
Every latent variable is governed by the equation

$$Z = \mathbf{x}_i^T \beta + \epsilon$$

If our model has J cathegories, we assume J-1 cathegory cutoffs, under the constraint

$$-\infty < \gamma_1 < ... < \gamma_{J-1} < +\infty$$

The relationship between Y_i and Z_i is such that then the latent variable falls between the cutoffs γ_{c-1} and γ_c then the observatio is classified in cathegory c. To ensure the parameters are identifiable we impose $\gamma_1 = 0$.



Let $\mathbf{p}_i = (p_{i1}, ..., p_{iC})$ denote the vector of probabilities associated with the assignment of the i-th item into the categories. Then the probability of observing the data $\mathbf{y}_i$ given the probability vector $\mathbf{p}_i$ is

$$P(\mathbf{y}|\mathbf{p}_i) \propto \prod_{i=1}^{n} p_{iy_i} \tag{1}$$

We substitute the value

$$p_{ij} = P(Y_i \leq j) = P(\gamma_{j-1} < Z_i \leq \gamma_j) = F(\gamma_j - \mathbf{x}_i^T\beta) - F(\gamma_{j-1} - \mathbf{x}_i^T\beta)$$

into (1), and we get the likelihood function for β:

$$L(\beta, \gamma) = \prod_{i=1}^{n}[F(\gamma_{y_i} - \mathbf{x}_i^T\beta) - F(\gamma_{y_i-1} - \mathbf{x}_i^T\beta)]$$

And if we parametrize the model using the latent Z we obtain

$$L(\beta, \gamma, \mathbf{Z}) = \prod_{i=1}^{n}[f(Z_i - \mathbf{x}_i^T\beta)I(\gamma_{y_i-1} < Z_i < \gamma_{y_i})] \tag{2}$$

where $I(.)$ is the indicator function and $F(.) = \Phi(.)$

4.2 GIBBS SAMPLING SCHEME

Here we describe the basic algorithm for ordinal probit models with the first cutoff parameter γ_i fixed at 0. Using the latent variable reoresentaton in (2), and letting $\Phi(.)$ describe the standard normal density, we get

$$g(\beta, \gamma, \mathbf{Z}|\mathbf{y}) \propto \prod_{i=1}^{n} [\Phi(Z_i - \mathbf{x}_i^T \beta) I(\gamma_{y_i - 1} < Z_i < \gamma_{y_i})]$$

which is a representation of the joint posterior density. A Gibbs Sampling for simulating from this joint posterior seems possible because the full conditional posterior distributions for $(\mathbf{Z}, \beta)$ are analytically available:

- $\beta|\mathbf{y}, \mathbf{Z}, \gamma \overset{ind}{\sim} N((\mathbf{X}^T\Omega^{-1}\mathbf{X})^{-1}(\mathbf{X}^T\Omega^{-1}\mathbf{Z}), (\mathbf{X}^T\Omega^{-1}\mathbf{X})^{-1})$
- $Z_i|\mathbf{y}, \beta, \gamma \overset{ind}{\sim} tN(\mathbf{x}_i^T\beta, 1)$ truncated at the current values of the category cutoffs
- the fully conditional distribution of γ_j given (β, y, Z) and $\gamma_k \neq j$ is proportional to:

$$\prod_{j=1}^{J} [1_{y_i=j} 1_{\gamma_{j-1} < Z_i < \gamma_j} + 1_{y_i=j} 1_{\gamma_j < Z_i < \gamma_{j+1}}]$$

which can be seen as uniform on the interval $(max(max(Z_i : Y_i = J), \gamma_{j-1}), min(min(Z_i : Y_i = j+1), \gamma_j))$

The Gibbs Sampler proceeds as follows: in each step we sample from $\beta|\mathbf{Z}, \gamma$ (because β s conditionally independent from $\mathbf{y}$ given z), and $z_i|\beta, \mathbf{y}, \gamma$. Start from $\beta^{(0)} = \hat{\beta}_{MLE}$ For $s = 1, ..., S$

1. sample $z^{(s)}$ from $p(z_i|\beta^{(0)}, \mathbf{y}, \gamma)$ full conditional of z
2. sample $\beta^{(s)}$ from $p(\beta|z^{(s-1)}, \gamma)$ full conditional of β

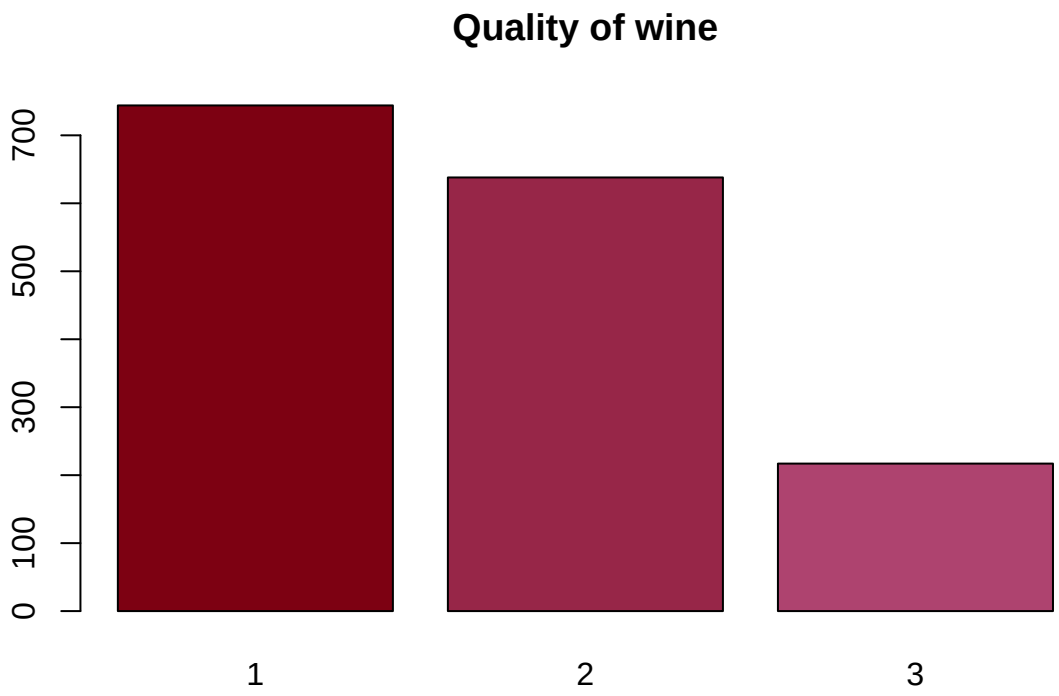
We now proceed to apply this sampling procedure to our dataset. We have already imported and cleaned the dataset, but this time we divide the response in three categories instead than two, with the following classification:

- Quality: **Bad - 1**
  - categories included: 3,4,5
- Quality: **Medium - 2**
  - categories included: 6
- Quality: **Good - 3**
  - categories included: 7,8

```
data <- read.csv("winequality-red.csv")
data$bound <- data$total.sulfur.dioxide - data$free.sulfur.dioxide
data2 <- subset(data, select = -c(pH, free.sulfur.dioxide, density, citric.acid,
                                  total.sulfur.dioxide))
data2$quality = as.factor(data2$quality)
levels(data2$quality) <- c(1,1,1,2,3,3)
```

And we visualize the categories to check there is a sufficient number of observations in each one:

```
my_col = hcl.colors(n = 10, palette = "Red-Purple")
Tab_prog = table(data2$quality)
barplot(Tab_prog, main = "Quality of wine", col = my_col)
```



Now we fit a generalized linear model to get the $\hat{\beta}_{MLE}$

```
mod2<-polr(data2$quality~., data=data2, method="probit")
sum_mod2<-summary(mod2)
```

```
##
## Ri-adattamento per ottenere Hessian
```

We then create the model matrix, the response vector, and the latent variable

```
y<-data2$quality             # response vector
n<-nrow(data2)
X<-subset(data2, select = -quality)
X<-as.matrix(X)              # model matrix
X.tmp = as.data.frame(X)
beta<-mod2$coefficients      # beta vector
eta <- X %*% beta
head(X.tmp)
```

```
##   fixed.acidity volatile.acidity residual.sugar chlorides sulphates alcohol
## 1           7.4             0.70            1.9     0.076      0.56     9.4
## 2           7.8             0.88            2.6     0.098      0.68     9.8
## 3           7.8             0.76            2.3     0.092      0.65     9.8
## 4          11.2             0.28            1.9     0.075      0.58     9.8
## 5           7.4             0.70            1.9     0.076      0.56     9.4
## 6           7.4             0.66            1.8     0.075      0.56     9.4
##   bound
## 1    23
## 2    42
```

```
## 3    39
## 4    43
## 5    23
## 6    27
```

4.2.1 POLYGIBBS PACKAGE

Our first approach is to carry out the parameter estimation through Gibbs Sampling, which is implemented in the package ***PolyGibbs***.

The function that returns the parameters and gammas' estimates is *MultinomGibbs_fit*, and requires a Train and Test set. We used an 80:20 ratio.

```
Train_ID = sample(1:nrow(X), round(nrow(X)*0.8), replace=FALSE) #Train Data IDS
Train_X = X[Train_ID, ]  # Train Data Covariates
Test_X = X[-Train_ID, ]  # Test Data Covariates
Train_Y = y[Train_ID]    # Train Data Response
Test_Y = y[-Train_ID]    # Test Data Response
K = 3
nIter = 1000
burn_in = 500
```

Now we apply the function and store the results.

```
invisible(capture.output(Result <- MultinomGibbs_fit(Train_X, Train_Y,
                                                     nIter, burn_in, K)))
list(estimates = Result$estimates,
     Train_Accuracy_tail = tail(Result$Train_Accuracy, n =10),
     beta_matrix_tail= tail(Result$beta_matrix, n = 10))

estimates = Result$estimates
gamma_estimates = Result$gamma_estimates
gamma_update = Result$gamma_update
beta_matrix = Result$beta_matrix
```

Once we have fitted the model, we need to test the predictive power of the model on a test set. We use the package function for prediction *MultinomGibbs_pred* and we store the results.

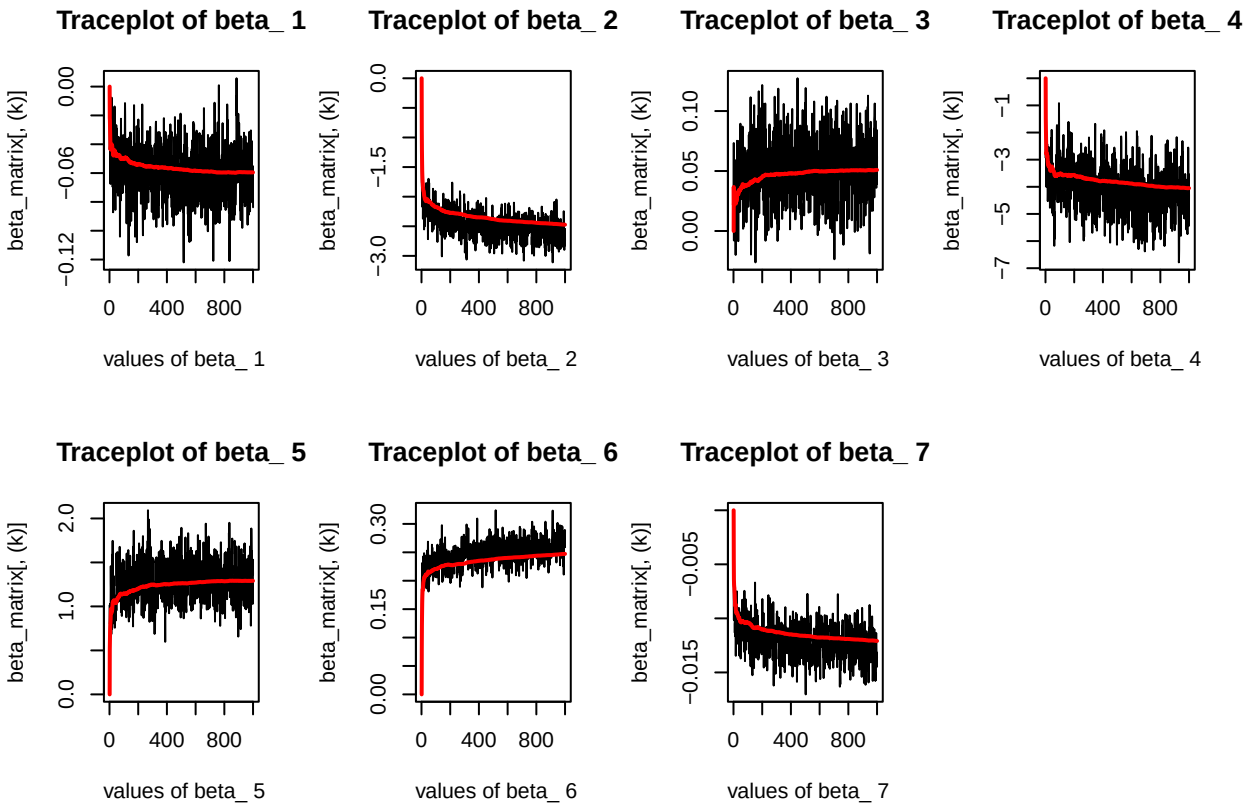
```
predicted_Y<-MultinomGibbs_pred(estimates, gamma_estimates,Test_X)
```

And we use another function to find the accuracy

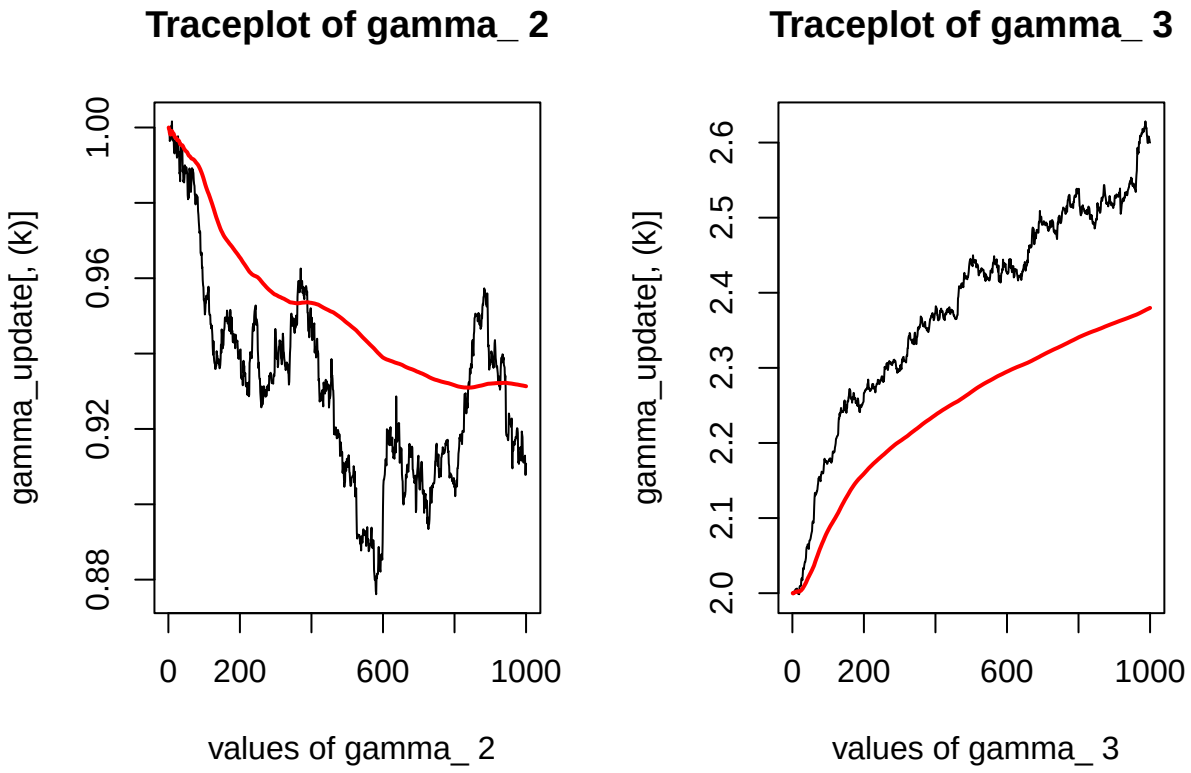
```
MultinomGibbs_Test_Accuracy(predicted_Y, Test_Y, K)
```

```
## [1] 57.8125
```

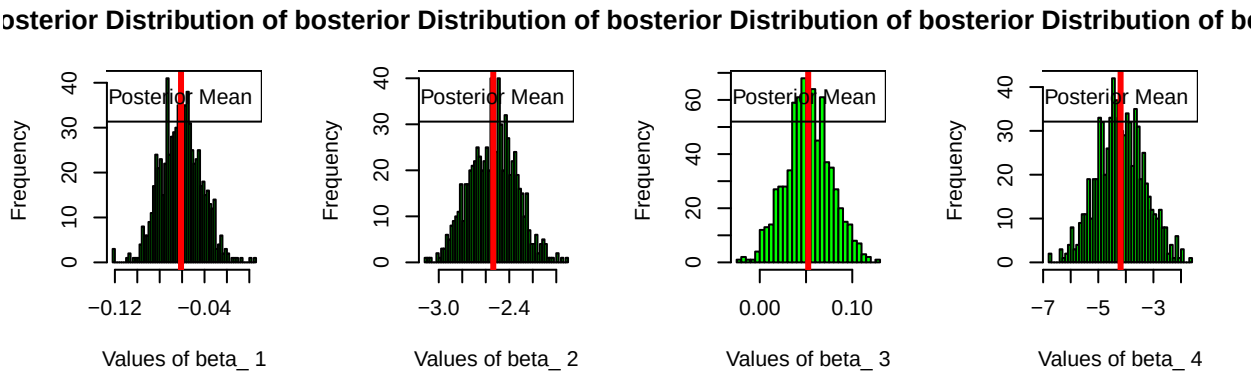
In order to study the convergence of the Gibbs Sampler algorithm we need to draw the **trace plot** of regression parameter estimates that plots the estimated posterior means of various parameters over all the iterations of the chain. We should expect the chain to more or less stabilize around the estimated posterior mean as we progress through the iterations.

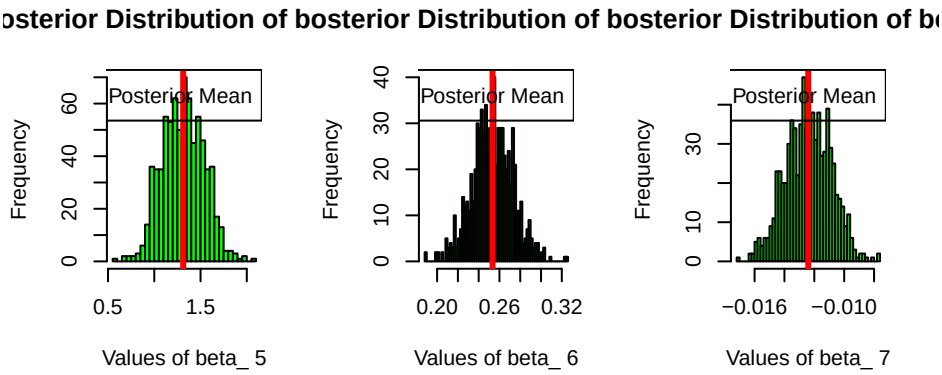


We also draw the trace plot of regression parameter estimates that plots the estimated posterior means of various parameters over all the iterations of the chain. We should expect the chain to more or less stabilize around the estimated posterior mean as we progress through the iterations.

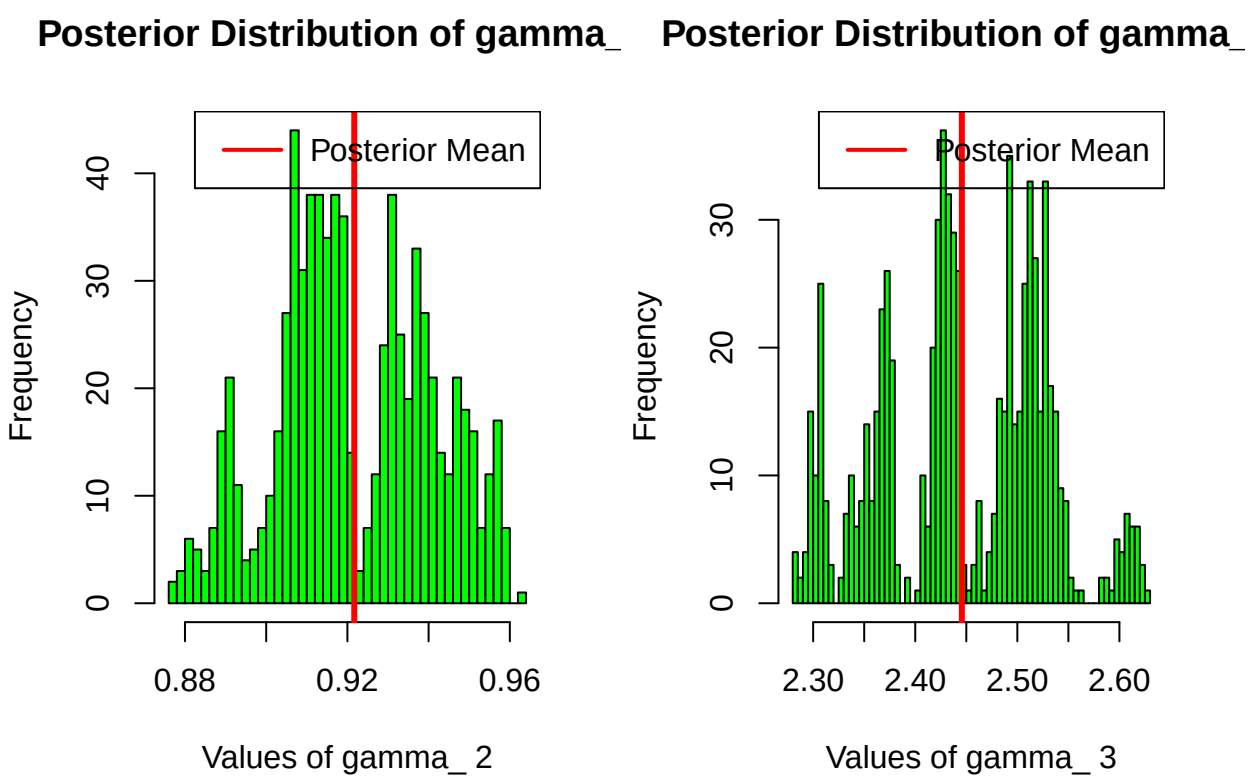


For studying the posterior distribution of the estimated regression parameters is neccessary to observe the dispersion of the drawn values around estimated posterior mean. We plot the posterior frequency distribution of the estimated parameters and their posterior mean.





We also plot the posterior frequency distribution of the estimated γs and their posterior mean.



We can see the β estimates look like they eventually converge, and the distribution of the drawn variables seems to be centered around the posterior mean. However we cannot say the same about the γ estimates: they do not seem to converge nor to be centered around the posterior mean. Another issue is that the accuracy is also not very high. In addition, the implementation of the algorithm is very slow and, for this reason, we can't set an adequate number of iterations.

For these reasons we tried to compute the β estimates by writing our own MCMC algorithm, which follows the same logic explained above.

4.2.2 GIBBS SAMPLER

We apply the same reasoning as in the binomial case and proceed to implement a Gibbs Sampling algorithm.

```
mode <- function(v) {
  uniqv <- unique(v)
  uniqv[which.max(tabulate(match(v, uniqv)))]
}

n = nrow(X)
p = ncol(X)
ns <- table(y)

# Priors
beta.0 <-  rep(0, p)                    # Prior Mean for Beta
V <- diag(0.01, p)                      # Prior Precision of Beta (vague)

# Initial Values
```

```r
gam1 <- polr(y ~ X, method = "probit")$zeta[1]
gam2<-polr(y ~ X, method = "probit")$zeta[2]
beta<-polr(y ~ X, method = "probit")$coefficients
z<-rep(0,n) # Latent Normal Variable
Z <- rep(0,n)
Y_hat_s<-rep(0,n)

# Create matrices to store results
nsim<-20000          # Number of MCMC Iterations
thin<-10                        # Thinning interval
burn<-5000           # Burnin
lastit<-(nsim-burn)/thin      # Last stored value
Beta <- matrix(NA, lastit, p)
Gam1<-rep(0,lastit)
Gam2<-rep(0,lastit)
Y_hat <- matrix(NA, lastit, n)

tmp<-proc.time()
for (i in 1:nsim) {

  # Draw latent z using inverse CDF method, which is faster than tnorm function here
  muz<-X%*%beta # mean of Z
  z[y == 1] <- rtruncnorm(ns[1], mean = muz[y == 1], sd = 1, a = -Inf, b = gam1)
  z[y == 2] <- rtruncnorm(ns[2], mean = muz[y == 2], sd = 1, a = gam1, b = gam2)
  z[y == 3] <- rtruncnorm(ns[3], mean = muz[y == 3], sd = 1, a = gam2, b = Inf)

  # Update gamma1
  gam1<-runif(1,max(z[y==1]),min(z[y==2]))

  # Update gamma2
  gam2<-runif(1,max(z[y==2]),min(z[y==3]))

  # Update beta
  V.n    = V + t(X)%*%X
  beta.n = solve(V + t(X)%*%X)%*%(t(X)%*%z + V%*%beta.0)
  beta = c(rmvnorm(1, beta.n, solve(V.n)))

  ##################
  # Prediction     #
  ##################
    Z <- rnorm(n, muz, 1)

  for (k in 1:n){
    if (Z[k] < gam1) {
      Y_hat_s[k] = 1
    }
    else if (Z[k] < gam2 & z[k] > gam1){
      Y_hat_s[k] = 2
    } else {
      Y_hat_s[k] = 3
    }
}
```

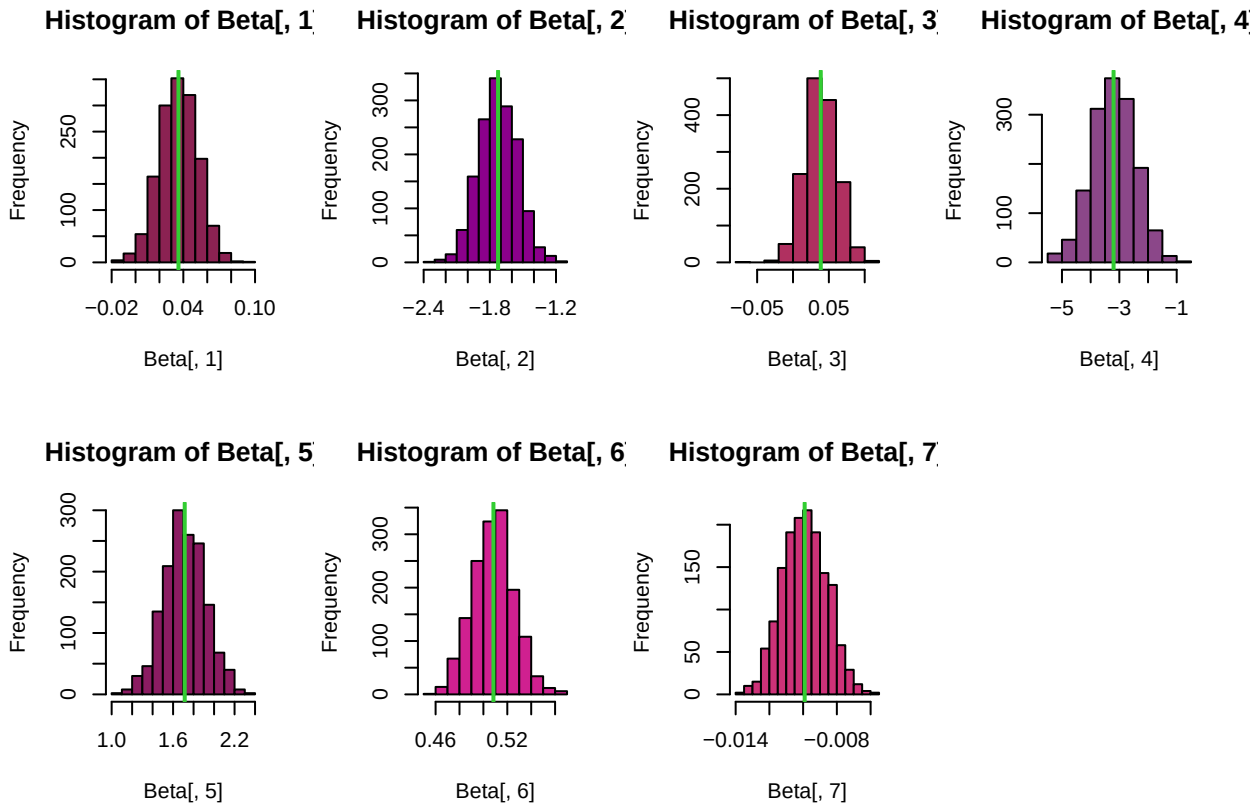
```
  ##################
  # Store Results #
  ##################
  if (i> burn & i%%thin==0) {
    j<-(i-burn)/thin
    Beta[j,] <-  beta
    Gam1[j]<-gam1
    Gam2[j]<-gam2
    Y_hat[j,] <- Y_hat_s
  }

  if(i%%100==0) print(i)
}

proc.time()-tmp # MCMC run time
```

```
############
# Results #
############
mbeta<-apply(Beta, 2, mean)
mgam1<-mean(Gam1)
mgam2<-mean(Gam2)
```

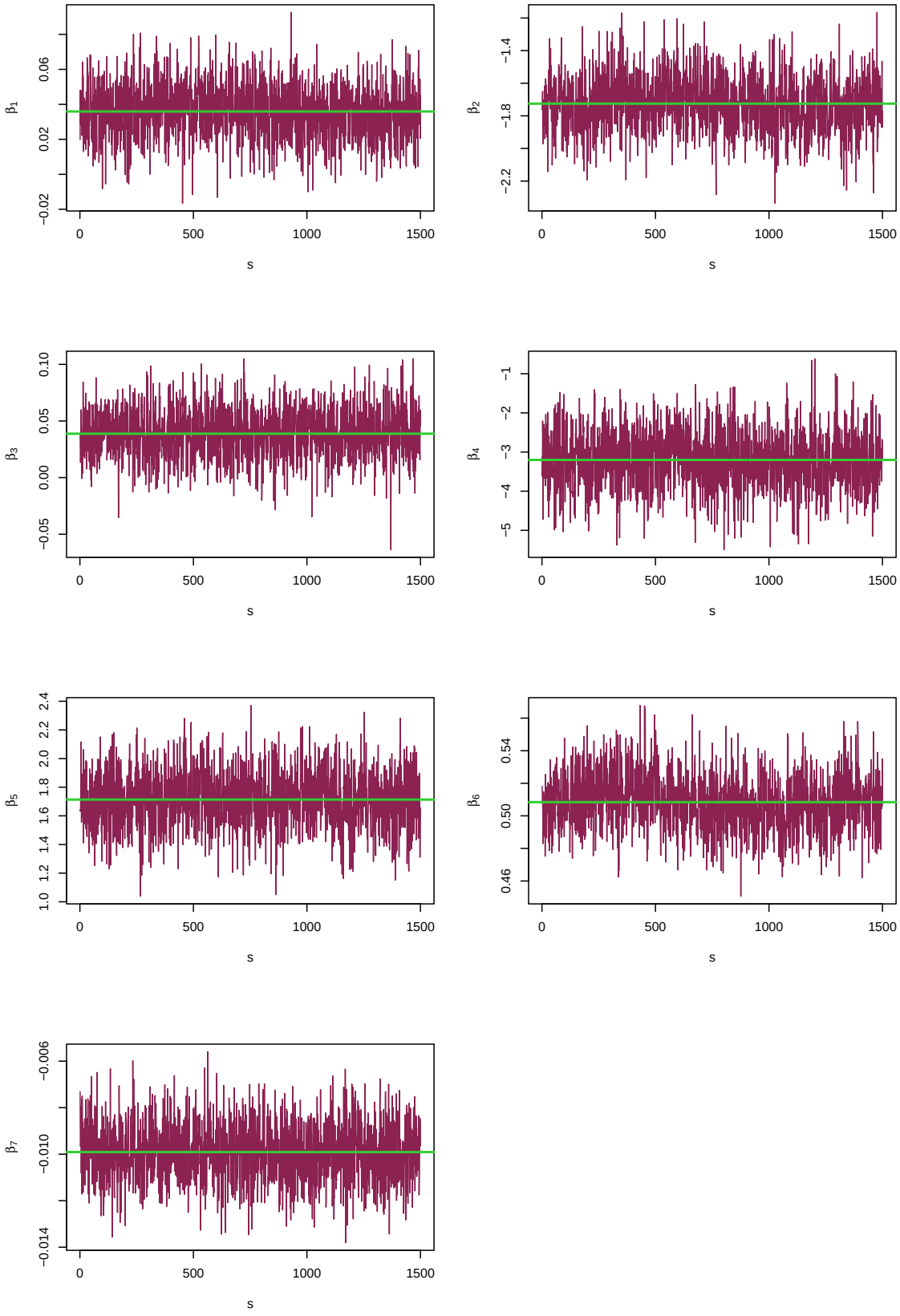
```
par(mfrow=c(2,4))
{hist(Beta[,1], col='violetred4')
abline(v=mbeta[1], col="limegreen", lwd=2)
hist(Beta[,2], col='magenta4')
abline(v=mbeta[2], col="limegreen", lwd=2)
hist(Beta[,3], col='maroon')
abline(v=mbeta[3], col="limegreen", lwd=2)
hist(Beta[,4], col='orchid4')
abline(v=mbeta[4], col="limegreen", lwd=2)
hist(Beta[,5], col='maroon4')
abline(v=mbeta[5], col="limegreen", lwd=2)
hist(Beta[,6], col='violetred')
abline(v=mbeta[6], col="limegreen", lwd=2)
hist(Beta[,7], col='violetred3')
abline(v=mbeta[7], col="limegreen", lwd=2)
}
```



4.2.3 MCMC DIAGNOSTIC

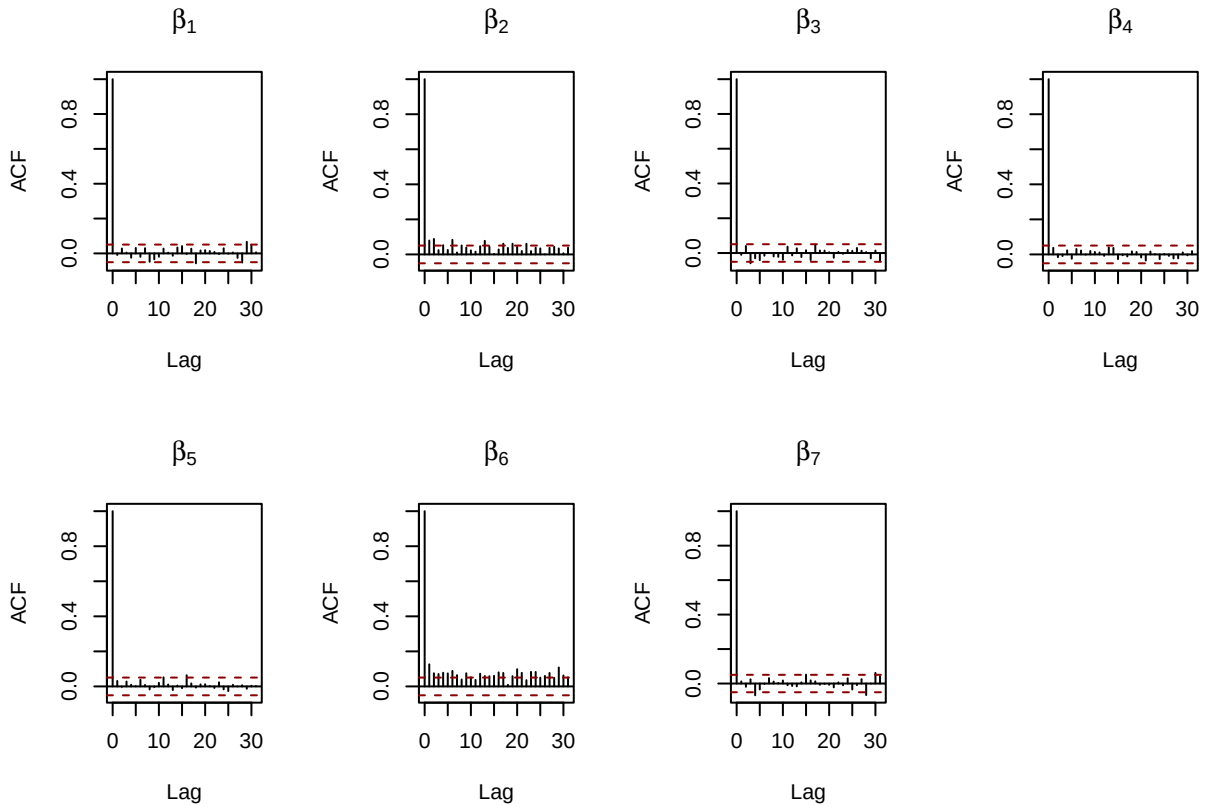
- GRAPHICAL CHECKS

First of all we use the ***trace plots*** of the sampled values $\beta_{\mathbf{i}}^{(\mathbf{s})}$ across the iterations $s = 1, ...S$. The chains should be concentrated around a region of high posterior probability and **no particular trends** should be observed in the chains.


β1
0.06
0.02
−0.02
β2
−1.4
−1.8
−2.2
β3
0.10
0.05
0.00
−0.05
β4
−1
−2
−3
−4
−5
β5
2.4
2.2
2.0
1.8
1.6
1.4
1.2
1.0
β6
0.54
0.50
0.46
β7
−0.006
−0.010
−0.014
0
500
1000
1500
s

We can't see any particular pattern in these trace plots.

We now use **acf** plots to check for autocorrelations in the chains. In fact, although the MCMC output is a dependent sequence from the posterior, its aim is to simulate random (and independent) draws from the posterior itself.



In this case, we don't see dependencies in any of the chains.

- FORMAL CONVERGENCE DIAGNOSTIC

As in the binomial case, we are going to perform some formal tests.

The first one is the *Geweke Test*. Again, we reject the null hypothesys of stationarity if $|Z_n|$ is "large". Therefore, if we fix $\alpha = 0.05$, with a value of $|Z_n| > 1.96$ we are in the rejection region of the null hypothesis.

	beta1	beta2	beta3	beta4	beta5	beta6	beta7
Z score	1.07187	1.17712	0.09858	-0.0132	-0.73749	0.58899	1.37375

Looking at these results, we can state that the Geweke Test indicates that the chains produced by our algorithm actually reached convergence.

We now evaluate the **Effective Sample Size (ESS)**

```
## in our case G is equal to:  1500
```

```
ESS <- effectiveSize(Beta)
```

```
ESS <- as.matrix(ESS)
colnames(ESS) <- 'ESS'
row.names(ESS) <- c('beta1',  'beta2', 'beta3', 'beta4', 'beta5', 'beta6', 'beta7')
```

	ESS
beta1	1500.0000
beta2	809.7577
beta3	1585.5358
beta4	1390.1082
beta5	1500.0000
beta6	218.5523
beta7	1657.3853

4.3. PREDICTION

Finally, we checked how well our algorithm performs. We added a step in the orignial algorithm: after updating the γs and β we sample from $Z_i \sim N(x_i^T\beta, 1)$ so that: * for each observation i, at time s we have: $\gamma_1^{(s)}, \gamma_2^{(s)}, \beta^{(s)}$ * we sample $Z_i^{(s)}$ from $N(\beta^{T(s)}\mathbf{x}_i, 1)$ * set

$$\begin{cases} y_i^{(s)} = 1 & if\ Z_i < \gamma_1 \\ y_i^{(s)} = 2 & if\ \gamma_1 < Z_i < \gamma_2 \\ y_i^{(s)} = 3 & if\ Z_i > \gamma_2 \end{cases}$$

- we obtain a (S, n) matrix, and we set $\hat{Y}_i$ as the mode of each column of such matrix
- we calculate the estimated probabilities as

$$\hat{P}(Y_i = 1) = \frac{1}{n}\sum_{i=1}^{N} \mathcal{I}(\hat{Y}_i = 1)$$

$$\hat{P}(Y_i = 2) = \frac{1}{n}\sum_{i=1}^{N} \mathcal{I}(\hat{Y}_i = 2)$$

$$\hat{P}(Y_i = 3) = \frac{1}{n}\sum_{i=1}^{N} \mathcal{I}(\hat{Y}_i = 3)$$

Finally, we compare these values with the observed proportions of each class of the response.

```
mode_ <- apply(Y_hat, 2, mode)

p_1_hat= (1/n)*length(which(mode_==1))
p_2_hat= (1/n)*length(which(mode_==2))
p_3_hat= (1/n)*length(which(mode_==3))


one<-table(y)[1]/n
two<-table(y)[2]/n
three<-table(y)[3]/n

d_multi <- tibble('target' = y,
                  'prediction' = mode_)

conf_mat <- confusion_matrix(targets=d_multi$target,
                             predictions = d_multi$prediction)

conf_mat$'Overall Accuracy'
```

```
## [1] 0.6285178
```

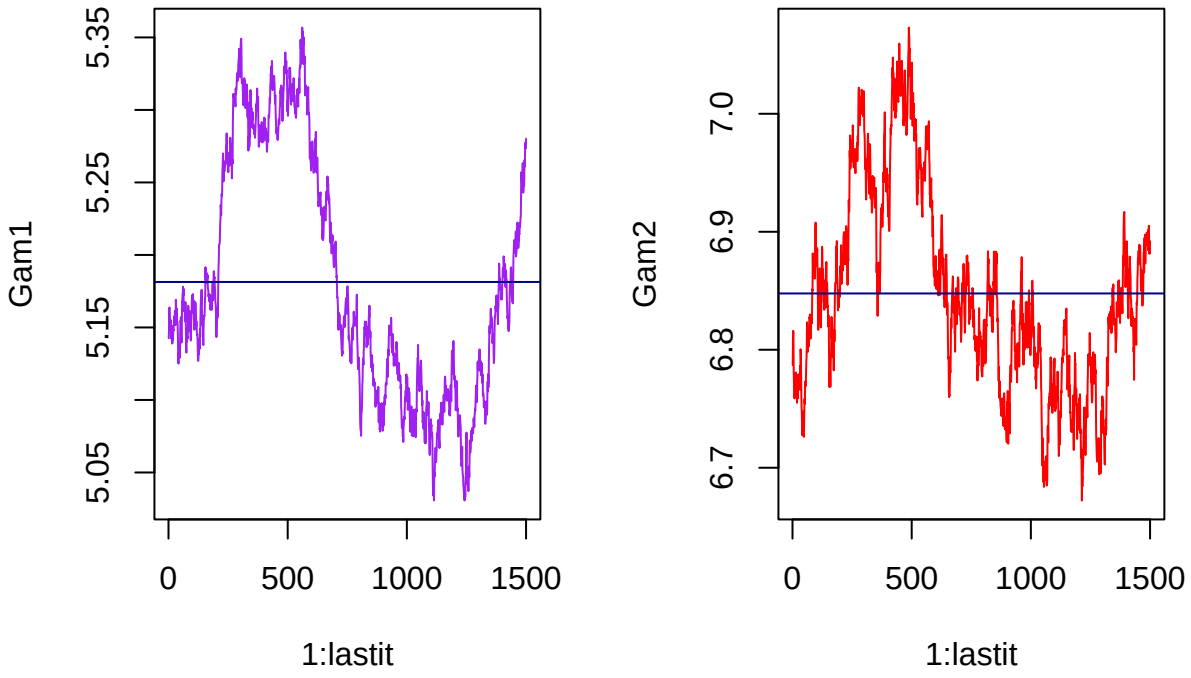
5. CONCLUSIONS

To sum up, carried out our analysis on two different models: a **binary model** and a **multinomial model**. For the multinomial model we were also able to R package "PolyGibbs", which implements a Gibbs Sampler algorithm. We reproduced a similar algorithm for the sake of comparison, and we added a step that we used for prediction. For both models we noticed that the MCMC chains of the predicted β seem to converge nicely. However, the precision we obtain with our algorithm is not very high, and the trace plots for the γs show problems with convergence.

```
par(mfrow=c(1,2))

plot(1:lastit,Gam1,type="l",col="purple")
abline(h=mgam1,col="blue4")

plot(1:lastit,Gam2,type="l",col="red")
abline(h=mgam2,col="blue4")
```



We could expect this result since the algorithm that we implemented - and that the PolyGibbs packages uses - is just a preliminary algorithm. A much more accurate estimation would be obtained by adding a Metropolis-Hastings step in the Gibbs Sampler. This step is used to sample from the conditional distribution of γ given y and β. However we chose to focus on a simpler approach as a first step to understand the theory behind an ordered multinomial probit regression.