Bayesian Logistic Regression

Alex B.Dunbar

8/9/2021

Clear environment, graphics and console

```
# Clear environment ####
rm(list = ls())

# Clear plots
graphics.off()  # Clears plots, closes all graphics devices

# Clear console
cat("\014")  # ctrl+L
```

Install required packages

1. Introduction

This is an implementation of Bayesian logistic regression using Metropolis-Hastings posterior sampling. The data is related to red variants of the Portuguese "Vinho Verde" wine. There are 11 regressor variables (covariates) which are based on physiochemical tests. The output variable is `good`, (1-good, 0-not-good) based on the `quality` variable which is a score between 0 and 10. `quality` is based on sensory data.

2. Import data.

The dataset is described in the publication by Cortez, P., Cerdeira, A., Almeida, F., Matos, T., & Reis, J. (2009). Modeling wine preferences by data mining from physicochemical properties..

```
wine <- read.csv("winequality-red.csv")
```

Check for missing values

There are no `NA` values in the dataset.

```
any(is.na(wine))
```

```
## [1] FALSE
```

3. Classify "good" wines

We want to implement a logistic regression, therefore we want a response variable which assumes values either 0 or 1. Suppose we consider "good" a wine with quality above 6.5 (included). We add a column called `good` that is `1` if the `quality` is greater than 6.5 and `0` if `quality` is less than 6.5. We also add a column of strings called `drinkable`, relating 1 and 0 to "good" and "bad" as appropriate.

```
wine %<>%
 mutate(good = ifelse(quality > 6, 1, 0)) %>%
 mutate(drinkable = ifelse(quality > 6, "good", "bad")) %>%
 mutate(quality = factor(quality, levels=1:10))
```

Remove spaces from names to make references easier.

```
colnames(wine) <- c("fixed_acidity", "volatile_acidity", "citric_acid",
                    "residual_sugar", "chlorides", "free_sulfur_dioxide",
                    "total_sulfur_dioxide", "density", "pH", "sulphates",
                    "alcohol", "quality", "good", "drinkable")
```

The structure of the dataset, `str(wine)`, will confirm that there are no `NA` values, the feature names, the feature type (num, Factor, chr) and the first 10 values.

```
str(wine)
```

```
## 'data.frame':    1599 obs. of  14 variables:
##  $ fixed_acidity       : num  7.4 7.8 7.8 11.2 7.4 7.4 7.9 7.3 7.8 7.5 ...
##  $ volatile_acidity    : num  0.7 0.88 0.76 0.28 0.7 0.66 0.6 0.65 0.58 0.5 ...
##  $ citric_acid         : num  0 0 0.04 0.56 0 0 0.06 0 0.02 0.36 ...
##  $ residual_sugar      : num  1.9 2.6 2.3 1.9 1.9 1.8 1.6 1.2 2 6.1 ...
##  $ chlorides           : num  0.076 0.098 0.092 0.075 0.076 0.075 0.069 0.065 0.073 0.071 ...
##  $ free_sulfur_dioxide : num  11 25 15 17 11 13 15 15 9 17 ...
##  $ total_sulfur_dioxide: num  34 67 54 60 34 40 59 21 18 102 ...
##  $ density             : num  0.998 0.997 0.997 0.998 0.998 ...
##  $ pH                  : num  3.51 3.2 3.26 3.16 3.51 3.51 3.3 3.39 3.36 3.35 ...
##  $ sulphates           : num  0.56 0.68 0.65 0.58 0.56 0.56 0.46 0.47 0.57 0.8 ...
##  $ alcohol             : num  9.4 9.8 9.8 9.8 9.4 9.4 9.4 10 9.5 10.5 ...
##  $ quality             : Factor w/ 10 levels "1","2","3","4",..: 5 5 5 6 5 5 5 7 7 5 ...
##  $ good                : num  0 0 0 0 0 0 0 1 1 0 ...
##  $ drinkable           : chr  "bad" "bad" "bad" "bad" ...
```

Overview of dataset

```
wine %>% skim()
```

Table 1: Data summary

Name	Piped data
Number of rows	1599
Number of columns	14

Column type frequency:	
character	1
factor	1
numeric	12
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
drinkable	0	1	3	4	0	2	0

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
quality	0	1	FALSE	6	5: 681, 6: 638, 7: 199, 4: 53

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
fixed_acidity	0	1	8.32	1.74	4.60	7.10	7.90	9.20	15.90	
volatile_acidity	0	1	0.53	0.18	0.12	0.39	0.52	0.64	1.58	
citric_acid	0	1	0.27	0.19	0.00	0.09	0.26	0.42	1.00	
residual_sugar	0	1	2.54	1.41	0.90	1.90	2.20	2.60	15.50	
chlorides	0	1	0.09	0.05	0.01	0.07	0.08	0.09	0.61	
free_sulfur_dioxide	0	1	15.87	10.46	1.00	7.00	14.00	21.00	72.00	
total_sulfur_dioxide	0	1	46.47	32.90	6.00	22.00	38.00	62.00	289.00	
density	0	1	1.00	0.00	0.99	1.00	1.00	1.00	1.00	
pH	0	1	3.31	0.15	2.74	3.21	3.31	3.40	4.01	
sulphates	0	1	0.66	0.17	0.33	0.55	0.62	0.73	2.00	
alcohol	0	1	10.42	1.07	8.40	9.50	10.20	11.10	14.90	
good	0	1	0.14	0.34	0.00	0.00	0.00	0.00	1.00	

Number of good and bad wines from each quality category

```
xtabs(~ good + quality, data=wine)
```

```
##     quality
## good   1   2   3   4   5   6   7   8   9  10
##    0   0   0  10  53 681 638   0   0   0   0
##    1   0   0   0   0   0   0 199  18   0   0
```

4. Frequentist Logistic Regression

We first implement a classic logistic regression model. From this model we can obtain the model coefficients. These coefficients are the MLE estimates which can be used as the initialisation values when we run the Metropolis-Hastings sampling. The classic logistic model is computed using the generalised linear model (glm) with `good` being a function of all other covariates except `quality` and `drinkable`.

```
fit <- glm(
  good ~ . - quality - drinkable,
  data = wine,
  family = binomial(link="logit")
)
```

Significant coefficients

The significant coefficients are labelled in the last column of the summary() function. Most significant coefficients are:

- `sulphates`($\Pr(\alpha > 6.924 = 4.39\text{e-}12)$), the `sulphates` coefficient is 3.750. A positive predictor whereby larger values of sulphate will predict a "good" wine.
- `alcohol`($\Pr(\alpha > 5.724 = 1.04\text{e-}08)$), the `alcohol` coefficient is 0.7533. A positive predictor whereby larger values of alcohol will predict a "good" wine.
- `volatile_acidity` ($\Pr(\alpha > |\text{-}3.291| = 9.99\text{e-}4)$), and the `volatile_acidity` coefficient is -2.581. A negative predictor whereby larger values of volatile acidity will predict a "not-good" wine.
- `total_sulfur_dioxide` ($\Pr(\alpha > |\text{-}3.378| = 7.31\text{e-}4)$) the `total_sulfur_dioxide` coefficient is -0.026. A negative predictor whereby larger values of total sulfur dioxide will predict a "not-good" wine.

```
fit %>% summary()
```

```
##
## Call:
## glm(formula = good ~ . - quality - drinkable, family = binomial(link = "logit"),
##     data = wine)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.9878  -0.4351  -0.2207  -0.1222   2.9869
##
## Coefficients:
##                        Estimate Std. Error z value Pr(>|z|)
## (Intercept)           2.428e+02  1.081e+02   2.247 0.024660 *
## fixed_acidity         2.750e-01  1.253e-01   2.195 0.028183 *
## volatile_acidity     -2.581e+00  7.843e-01  -3.291 0.000999 ***
## citric_acid           5.678e-01  8.385e-01   0.677 0.498313
## residual_sugar        2.395e-01  7.373e-02   3.248 0.001163 **
## chlorides            -8.816e+00  3.365e+00  -2.620 0.008788 **
## free_sulfur_dioxide   1.082e-02  1.223e-02   0.884 0.376469
## total_sulfur_dioxide -1.653e-02  4.894e-03  -3.378 0.000731 ***
## density              -2.578e+02  1.104e+02  -2.335 0.019536 *
## pH                    2.242e-01  9.984e-01   0.225 0.822327
## sulphates             3.750e+00  5.416e-01   6.924 4.39e-12 ***
## alcohol               7.533e-01  1.316e-01   5.724 1.04e-08 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1269.92  on 1598  degrees of freedom
## Residual deviance:  870.86  on 1587  degrees of freedom
## AIC: 894.86
##
## Number of Fisher Scoring iterations: 6
```

5. Impact on outcome by varying total_sulfur_dioxide

By fixing all coefficients and varying one, we can see the impact of one particular coefficient has on the probabilistic outcome of the logistic regression.

As a comparison I have fixed all significant coefficients in turn (`total_sulfur_dioxide`, `volatile_acidity`, `sulphates` and `alcohol`)

First, we save coefficients to their own variable.

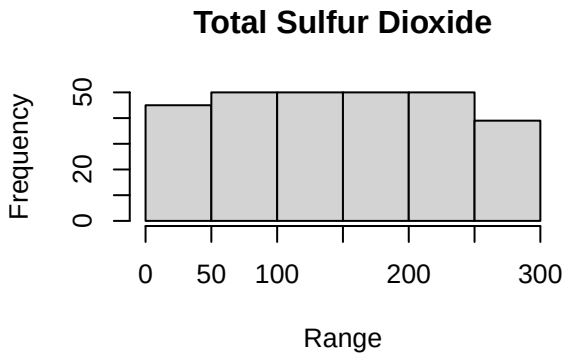
```
b0 <- fit$coefficients[1]  # Intercept = 242.76251933
b1 <- fit$coefficients[2]  # fixed_acidity = 0.27495289
b2 <- fit$coefficients[3]  # volatile_acidity = -2.58100211
b3 <- fit$coefficients[4]  # citric_acid = 0.56779433
b4 <- fit$coefficients[5]  # residual_sugar = 0.23946420
b5 <- fit$coefficients[6]  # chlorides = -8.81636544
b6 <- fit$coefficients[7]  # free_sulfur_dioxide = 0.01082060
b7 <- fit$coefficients[8]  # total_sulfur_dioxide = -0.01653061
b8 <- fit$coefficients[9]  # density = -257.79757874
b9 <- fit$coefficients[10]  # pH = 0.22418522
b10 <- fit$coefficients[11]  # sulphates = 3.74987886
b11 <- fit$coefficients[12]  # alcohol = 0.75333905
```

Then save each individual mean values

```
fixed_acidity_mean <- mean(wine$fixed_acidity)
volatile_acidity_mean <- mean(wine$volatile_acidity)
citric_acid_mean <- mean(wine$citric_acid)
residual_sugar_mean <- mean(wine$residual_sugar)
chlorides_mean <- mean(wine$chlorides)
free_sulfur_dioxide_mean <- mean(wine$free_sulfur_dioxide)
total_sulfur_dioxide_mean <- mean(wine$total_sulfur_dioxide)
density_mean <- mean(wine$density)
pH_mean <- mean(wine$pH)
sulphates_mean <- mean(wine$sulphates)
alcohol_mean <- mean(wine$alcohol)
```

Compute the range of each significant coefficient, `total_sulfur_dioxide`, `sulphates`, `alcohol`, and `volatile_acidity`

```
total_sulfur_dioxide_range <- seq(from=min(wine$total_sulfur_dioxide), to=max(wine$total_sulfur_dioxide
sulphates_range <- seq(from=min(wine$sulphates), to=max(wine$sulphates), by=0.002)
alcohol_range <- seq(from=min(wine$alcohol), to=max(wine$alcohol), by=0.05)
volatile_acidity_range <- seq(from=min(wine$volatile_acidity), to=max(wine$volatile_acidity), by=0.002)
```



Plot of the range histogram of each of the four most significant covariates.

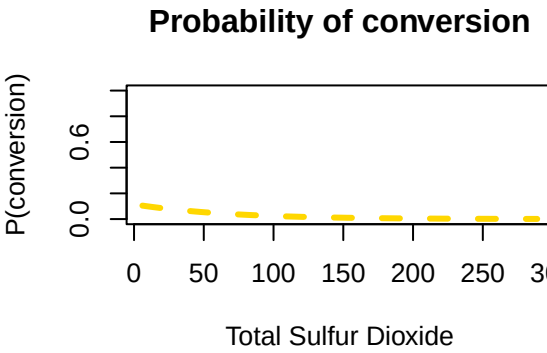
Calculate probabilities for each significant coefficient

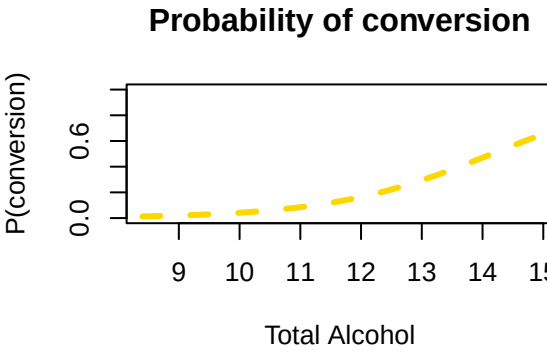
```
total_sulfur_dioxide_GOOD <- b0 + b1*fixed_acidity_mean + b2*volatile_acidity_mean +
  b3*citric_acid_mean + b4*residual_sugar_mean + b5*chlorides_mean + b6*free_sulfur_dioxide_mean +
  b7*total_sulfur_dioxide_range + b8*density_mean + b9*pH_mean + b10*sulphates_mean + b11*alcohol_mean

sulphates_GOOD <- b0 + b1*fixed_acidity_mean + b2*volatile_acidity_mean +
  b3*citric_acid_mean + b4*residual_sugar_mean + b5*chlorides_mean + b6*free_sulfur_dioxide_mean +
  b7*total_sulfur_dioxide_mean + b8*density_mean + b9*pH_mean + b10*sulphates_range + b11*alcohol_mean

alcohol_GOOD <- b0 + b1*fixed_acidity_mean + b2*volatile_acidity_mean +
  b3*citric_acid_mean + b4*residual_sugar_mean + b5*chlorides_mean + b6*free_sulfur_dioxide_mean +
  b7*total_sulfur_dioxide_mean + b8*density_mean + b9*pH_mean + b10*sulphates_mean + b11*alcohol_range

volatile_acidity_GOOD <- b0 + b1*fixed_acidity_mean + b2*volatile_acidity_range +
  b3*citric_acid_mean + b4*residual_sugar_mean + b5*chlorides_mean + b6*free_sulfur_dioxide_mean +
  b7*total_sulfur_dioxide_mean + b8*density_mean + b9*pH_mean + b10*sulphates_mean + b11*alcohol_mean
```

Calculate log odds probabilities for each

```
total_sulfur_dioxide_probs <- exp(total_sulfur_dioxide_GOOD)/(1 + exp(total_sulfur_dioxide_GOOD))
sulphates_probs <- exp(sulphates_GOOD) / (1 + exp(sulphates_GOOD))
alcohol_probs <- exp(alcohol_GOOD) / (1 + exp(alcohol_GOOD))
volatile_acidity_probs <- exp(volatile_acidity_GOOD) / (1 + exp(volatile_acidity_GOOD))
```

Plot the results.





The plots of each of the four covariates are the probabilities across the range of

6. Bayesian Logistic Regression

The following is a Bayesian logistic regression analysis. To perform a Bayesian analysis via Metropolis-Hastings algorithm we write a function defining the target distribution of the Beta coefficients. We work in terms of log posterior to avoid numerical problems with computer computation.

Likelihood.

The likelihood is a product of Bernoulli likelihoods for each combination of covariate and response variable for the i-th term.

$$L(\beta; y, x) = \prod_{i=1}^{n} \left(\frac{\exp(x_i\beta)}{1 + \exp(x_i\beta)} \right)^{y_i} \left(1 - \frac{\exp(x_i\beta)}{1 + \exp(x_i\beta)} \right)^{1-y_i}$$

Log-likelihood

$$\log L(\beta; y, x) = \sum_{i=1}^{n} y_i(x_i\beta) - \log[1 + \exp(x_i\beta)]$$

Prior

Independent normal prior distributions for each $\beta_j, j = 0, 1, \ldots, k$, such that $\beta_j \sim N(0, 100)$

Log posterior distribution

This function will compute the log-likelihood and the log-prior and return the log-posterior. The function is called in the Metropolis-Hastings algorithm to compute the distribution for the proposed value (β^*) for the next value the chain to compare against the existing value in the chain.

```
lpost.LR <- function(beta, x, y) {
  # beta: vector of coefficients
  # x: covariates (regressors)
  # y: response variable
  # ------------------------------
  # eta is the matrix product of the covariates and the coefficients
  eta <- as.numeric(x %*% beta)
  # logp: y=1, in terms of logistic function
  logp <- eta - log(1 + exp(eta))  # in log scale
  # logq: y=0, expressed in terms of logistic function
  logq <- log(1 - exp(logp))
  # sum of both contributions: when y=1 + when y=0
  logl <- sum(logp[y==1]) + sum(logq[y==0])
  # log prior: normal indep prior dists $N(0, 100)$
  lprior <- sum(dnorm(beta, 0, 100,log = T))  # 10: mixed, 50 var, 100 large var, 20 like 10, 5 low
  # return log posterior (likelihood + prior)
  return(logl + lprior)
}
```

Set seed for reproducible results

```
set.seed(1234)
```

Number of simulations

```
S <- 20000
```

Set `X` and `y` variables

```
X = cbind(rep(1, nrow(wine)), wine$fixed_acidity, wine$volatile_acidity, wine$citric_acid,
          wine$residual_sugar, wine$chlorides, wine$free_sulfur_dioxide, wine$total_sulfur_dioxide,
          wine$density, wine$pH, wine$sulphates, wine$alcohol)

y <- wine$good[1]
```

Initialisations for the coefficients

The first initialisation contains the MLE estimates for each coefficient. The remaining three initialisations are randomly selected values between 100 and 200 from the uniform distribution $runif(48, 100, 200)$.

```
# init <- matrix(data=c(runif(20, min = 100, max = 200)), nrow=4, ncol=5, byrow = T)  # for testing
init <- matrix(data=c(runif(48, min = 100, max = 200)), nrow=4, ncol=12, byrow = T)
```

Run a Metropolis-Hastings algorithm

```
# First initialisation
beta_mat_init1 <- matrix(NA, nrow = S, ncol = ncol(X))
k <- ncol(beta_mat_init1)
beta_mat_init1[1,] <- init[1,]
#beta_mat_init1[1,] <- as.numeric(coefficients(fit))  # initialise with MLE of each coefficient
acc <- 0
for (iter in 2:S) {
  # simulate all (k) values using previous value of beta as mean and set sd
  beta_star <- rnorm(k, beta_mat_init1[iter-1,], 5)
  # compute target distribution for proposed value
  newpost = lpost.LR(beta_star, X, y)
  # compute target distribution for old value
  oldpost = lpost.LR(beta_mat_init1[iter-1,], X, y)  # symmetric dist => no ratio computed

  # acceptance step, in log scale
  if (runif(1,0,1) > exp(newpost - oldpost)) {
    # chain doesn't move
    beta_mat_init1[iter,] = beta_mat_init1[iter-1,]
  } else {
    # add to chain and add 1 to counter
    beta_mat_init1[iter,] = beta_star
    acc=acc + 1
  }
  #if (iter%%1000 == 0) {print(c(iter, acc/iter))}
}
print(c(iter, acc/iter))
```

```
## [1] 2.0000e+04 8.9675e-01
```

```
# Second initialisation
beta_mat_init2 <- matrix(NA, nrow = S, ncol = ncol(X))
k <- ncol(beta_mat_init2)
beta_mat_init2[1,] <- init[2,]
acc <- 0
for(iter in 2:S){
  beta_star <- rnorm(k, beta_mat_init2[iter-1,], 5)
  newpost = lpost.LR(beta_star, X, y)
  oldpost = lpost.LR(beta_mat_init2[iter-1,], X, y)
  if (runif(1,0,1) > exp(newpost - oldpost)) {
    beta_mat_init2[iter,] = beta_mat_init2[iter-1,]
  } else {
    beta_mat_init2[iter,] = beta_star
    acc=acc + 1
  }
  #if (iter%%1000==0) {print(c(iter,acc/iter))}
```

```
}
print(c(iter, acc/iter))
```

```
## [1] 2.0000e+04 8.9945e-01
```

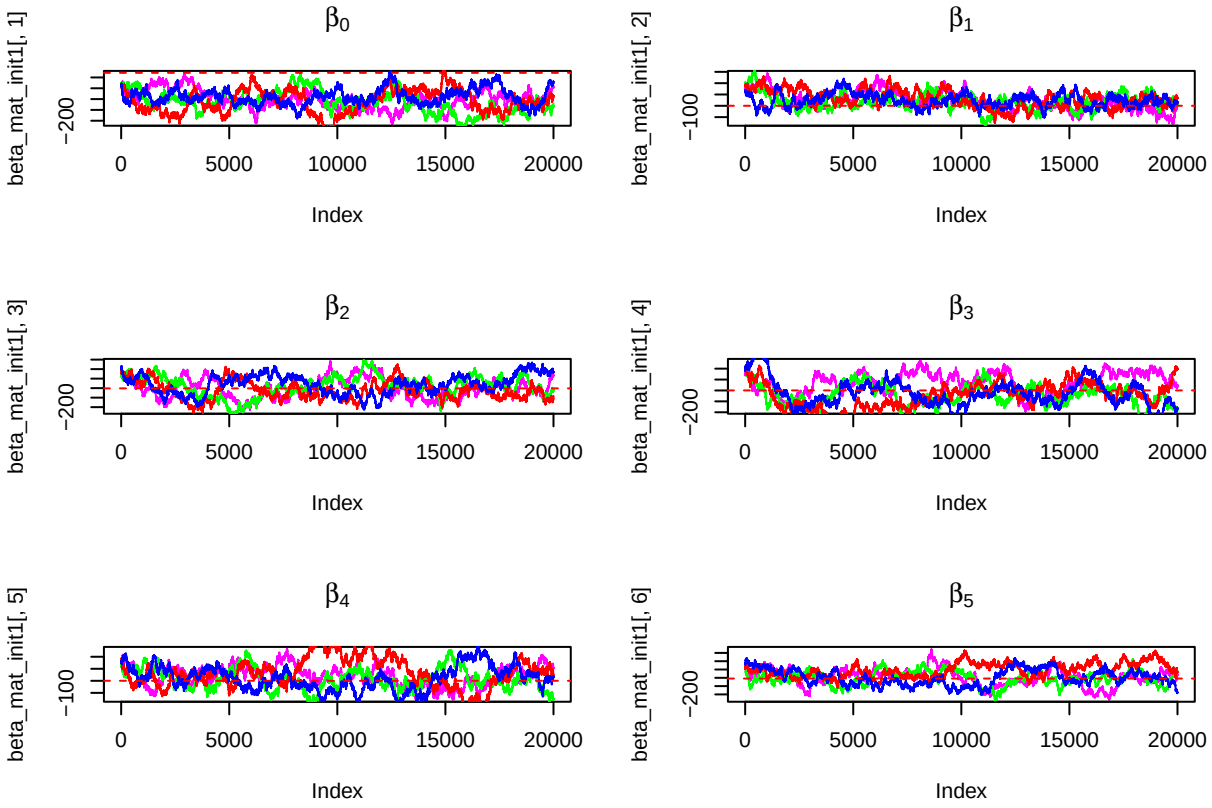
```
# Third initialisation
beta_mat_init3 <- matrix(NA,nrow=S,ncol=ncol(X))
k <- ncol(beta_mat_init3)
beta_mat_init3[1,] <- init[3,]
acc <- 0
for(iter in 2:S){
  beta_star <- rnorm(k,beta_mat_init3[iter-1,], 5)
  newpost=lpost.LR(beta_star,X,y)
  oldpost=lpost.LR(beta_mat_init3[iter-1,],X,y)
  if(runif(1,0,1)>exp(newpost-oldpost)){
    beta_mat_init3[iter,]=beta_mat_init3[iter-1,]
  } else{
    beta_mat_init3[iter,]=beta_star
    acc=acc+1
  }
  #if(iter%%1000==0){print(c(iter,acc/iter))}
}
print(c(iter, acc/iter))
```

```
## [1] 2.000e+04 9.046e-01
```

```
# Forth initialisation
beta_mat_init4 <- matrix(NA,nrow=S,ncol=ncol(X))
k <- ncol(beta_mat_init4)
beta_mat_init4[1,] <- init[4,]
acc <- 0
for(iter in 2:S){
  beta_star <- rnorm(k,beta_mat_init4[iter-1,], 5)
  newpost=lpost.LR(beta_star,X,y)
  oldpost=lpost.LR(beta_mat_init4[iter-1,],X,y)
  if(runif(1,0,1)>exp(newpost-oldpost)){
    beta_mat_init4[iter,]=beta_mat_init4[iter-1,]
  } else{
    beta_mat_init4[iter,]=beta_star
    acc=acc+1
  }
  #if(iter%%1000==0){print(c(iter,acc/iter))}
}
print(c(iter, acc/iter))
```

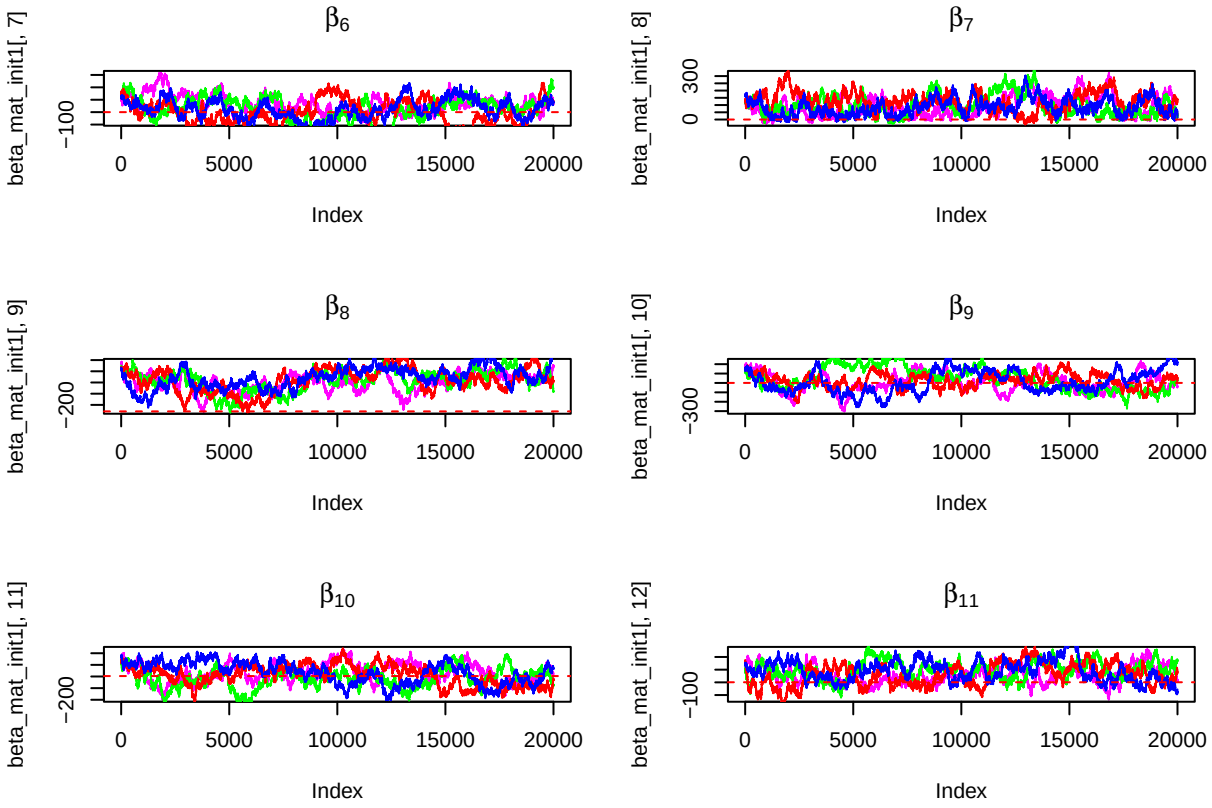
```
## [1] 2.0000e+04 8.9605e-01
```

Plot the chains for each coefficient (same plot)




```
fit$coefficients[1:6]
```

```
##      (Intercept)    fixed_acidity volatile_acidity      citric_acid
##      242.7625193        0.2749529       -2.5810021        0.5677943
##   residual_sugar        chlorides
##        0.2394642       -8.8163654
```



Comment

```
fit$coefficients[7:12]
```

```
##  free_sulfur_dioxide total_sulfur_dioxide             density
##           0.01082060          -0.01653061       -257.79757874
##                   pH            sulphates             alcohol
##           0.22418522            3.74987886           0.75333905
```

β_0 (`Intercept`: 242.7625): No convergence with MLE. All four chains do converge to a value around zero. β_1 (`fixed_acidity`: 0.275): First chain containing MLE converges to a value close but slightly under the coefficient. The other three chains containing random initialisations converge to a value approximately 20-30. β_2 (`volative_acidity`: -2.581): All four chains converge to a value close to the coefficient. β_3 (`citric_acid`: 0.568): All four chains converge to a value close to the coefficient. β_4 (`residual_sugar`: 0.239): All four chains converge to a value close to the coefficient. β_5 (`chlorides`: -8.816): All four chains converge to a value close to the coefficient. β_6 (`free_sulfur_dioxide`: 0.011): First chain containing MLE converges to a value close but slightly under the coefficient. The other three chains containing random initialisations converge to a value approximately 20-30. β_7 (`total_sulfur_dioxide`: -0.017): First chain converges to a value just below coefficient but chains 2-4 don't converge to coefficient. β_8 (`density`: -257.798): First chain converges to a value at about the coefficient but chains 2-4 converge to a value slightly above the coefficient β_9 (`pH`: 0.224): All four chains converge to a value close to the coefficient. β_{10} (`sulphates`: 3.750): All four chains converge to a value close to the coefficient. β_{11} (`alcohol`: 0.753): First chain converges to a value at about the coefficient but chains 2-4 converge to a value above the coefficient

Comment

How good is the distribution for the parameter?

- varied standard deviation of the prior in the posterior distribution function (0.5, 1, 5, 10, 20, 50) > increasing the standard deviation increases the instability > decreasing the standard deviation decreases ability to find convergence with "true" value
- varied standard deviation of the Metropolis-Hastings algorithm (0.5, 1, 5, 10, 20, 50)
- varied the random initialised values (+/- 100, 0-1, 0-10, 0-100, 10-100, 100-200, 500-1000) > some ranges close to zero produce errors

7. Posterior Predictive Distribution

Approximate the posterior predictive distribution of an unobserved variable characterised by the following values for each covariate:

```
fixed_acidity <- 7.5
volatile_acidity <- 0.6
citric_acid <- 0.0
residual_sugar <- 1.7
chlorides <- 0.085
free_sulfur_dioxide <- 5
total_sulfur_dioxide <- 45
density <- 0.9965
pH <- 3.4
sulphates <- 0.63
alcohol <- 12
```

```
S <- 20000
beta_mat2 <- matrix(NA, nrow = S, ncol = ncol(X))
beta_mat2[1,] <- as.numeric(coefficients(fit))

y_new <- c(1)
x_new <- c(1, fixed_acidity, volatile_acidity, citric_acid, residual_sugar,
           chlorides, free_sulfur_dioxide, total_sulfur_dioxide, density,
           pH, sulphates, alcohol)
```

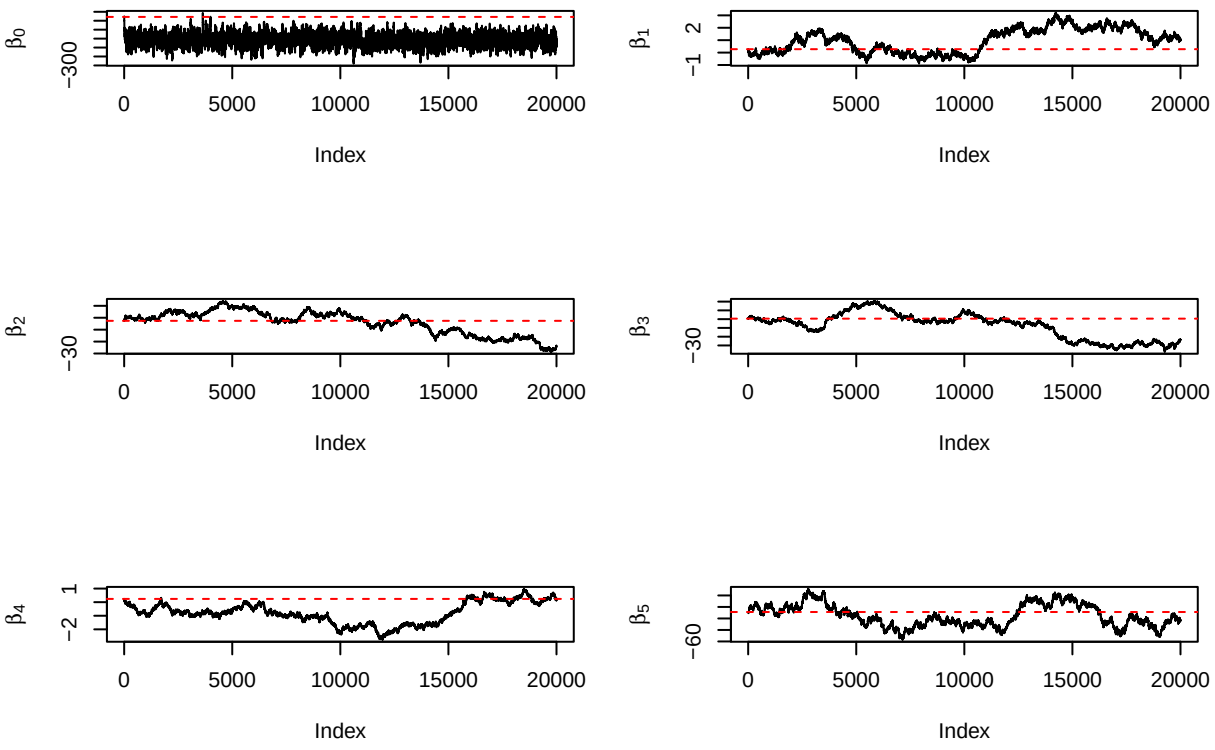
new model

```
library(mvtnorm)

# prediction

Omega_prop <- solve(t(X) %*% X)
k <- ncol(beta_mat2)
acc <- 0
for(iter in 2:S)
{
  # 1. Propose a new set of values
  beta_star <- rmvnorm(1, beta_mat2[iter-1,], 1.5 * Omega_prop)
```

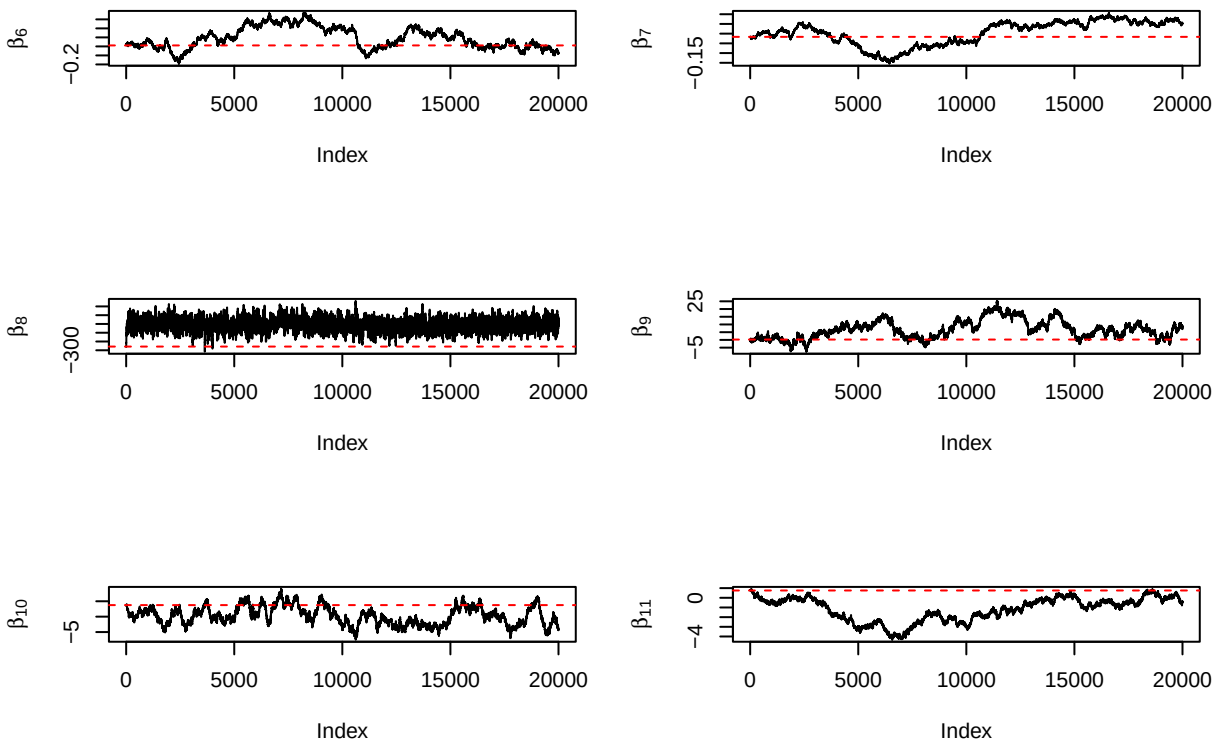
```
  # 2. Compute the posterior density on the proposed value and on the old value
  newpost=lpost.LR(t(beta_star), X, y)
  oldpost=lpost.LR(matrix(beta_mat2[iter-1,], ncol=1), X, y)

  # 3. Acceptance step
  if (runif(1, 0, 1) > exp(newpost - oldpost)) {
    beta_mat2[iter,] = beta_mat2[iter-1,]
  } else {
    beta_mat2[iter,] = beta_star
    acc = acc + 1
  }
  # 4. Print the stage of the chain
  if (iter%%1000 == 0){ print(c(iter, acc/iter)) }

  # 5. Prediction
  p_new <- exp(sum(beta_mat2[iter,] * x_new) ) / (1 + exp(sum(beta_mat2[iter,] * x_new) ))
  y_new[iter] <- rbinom(1,1,prob=p_new)
}
```

```
## [1] 1000.000    0.728
## [1] 2000.000    0.764
## [1] 3000.0000000    0.7856667
## [1] 4000.00000    0.78825
## [1] 5000.0000    0.7898
## [1] 6000.0000000    0.7936667
## [1] 7000.0000000    0.7944286
## [1] 8000.00000    0.79675
## [1] 9000.0000000    0.8001111
## [1] 1.00e+04 8.01e-01
## [1] 1.100000e+04 8.010909e-01
## [1] 1.200000e+04 8.026667e-01
## [1] 1.30e+04 8.03e-01
## [1] 1.400000e+04 8.033571e-01
## [1] 1.500000e+04 8.036667e-01
## [1] 1.60000e+04 8.04375e-01
## [1] 1.700000e+04 8.041765e-01
## [1] 1.800000e+04 8.050556e-01
## [1] 1.900000e+04 8.052632e-01
## [1] 2.000e+04 8.066e-01
```

Plots


β0
−300
β1
−1
2
β2
−30
β3
−30
β4
−2
1
β5
−60
0
5000
10000
15000
20000
Index

Plots



β_0 (`Intercept`: 242.7625): ## No convergence with MLE. converge to a value around zero. β_1 (`fixed_acidity`: 0.275): ##
β_2 (`volative_acidity`: -2.581): ## β_3 (`citric_acid`: 0.568): ## β_4 (`residual_sugar`: 0.239): ## β_5 (`chlorides`: -8.816): ## β_6 (`free_sulfur_dioxide`: 0.011): ## β_7 (`total_sulfur_dioxide`: -0.017): ## β_8 (`density`: -257.798): ## β_9 (`pH`: 0.224): ## β_{10} (`sulphates`: 3.750): ## β_{11} (`alcohol`: 0.753): ##

```
table(y_new[15000:20000])
```

```
##
##    0
## 5001
```

8. metrop() analysis of Q6

Implementation of the `metrop()` function was through the MCMC Package Example, Charles J. Geyer

```
library(mcmc)
out <- glm(good ~ . - quality - drinkable, data=wine, family=binomial, x=TRUE)
```

```
lupost_factory <- function(x, y) function(beta) {
  eta <- as.numeric(x %*% beta)
  logp <- ifelse(eta < 0, eta - log1p(exp(eta)), - log1p(exp(- eta)))
  logq <- ifelse(eta < 0, - log1p(exp(eta)), - eta - log1p(exp(- eta)))
  logl <- sum(logp[y == 1]) + sum(logq[y == 0])
  return(logl - sum(beta^2) / 8)
}
lupost <- lupost_factory(out$x, out$y)
```

```
set.seed(1234)
beta.init <- as.numeric(coefficients(out))
out <- metrop(lupost, beta.init, 1e3)

names(out)
```

```
##  [1] "accept"       "batch"        "initial"      "final"        "accept.batch"
##  [6] "initial.seed" "final.seed"   "time"         "lud"          "nbatch"
## [11] "blen"         "nspac"        "scale"        "debug"
```

Look at the acceptance rate.

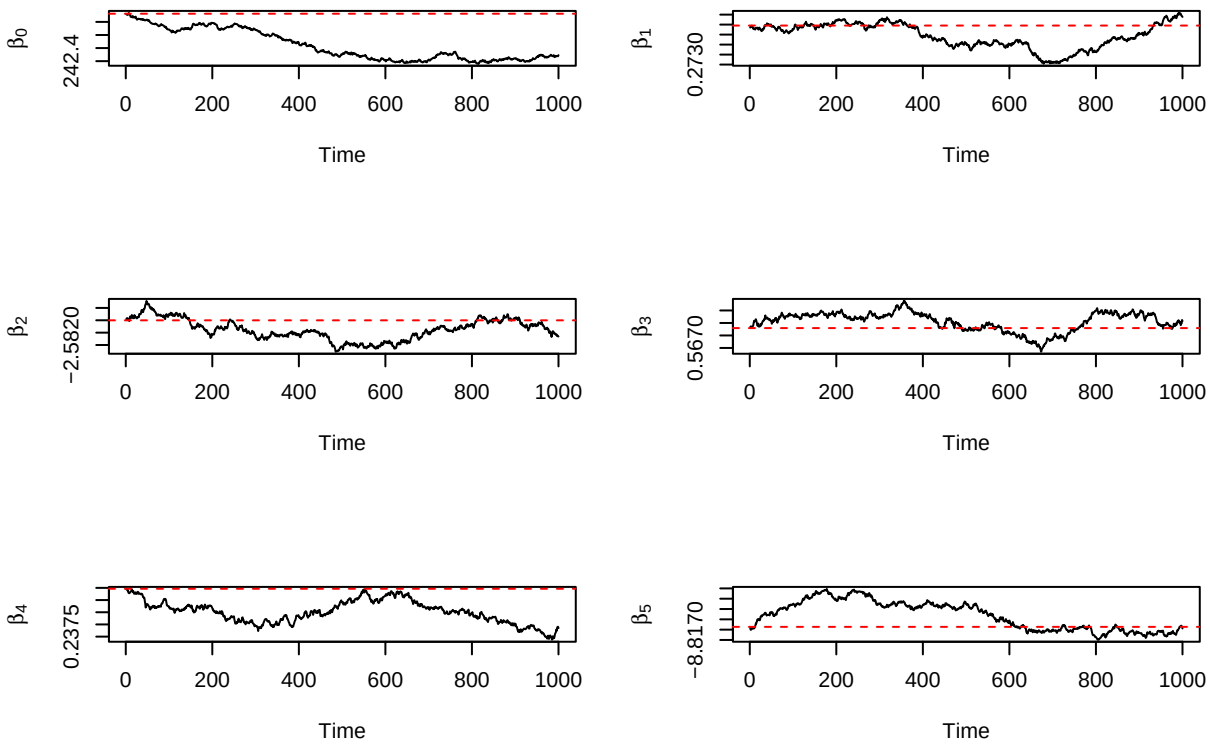
```
out$accept
```

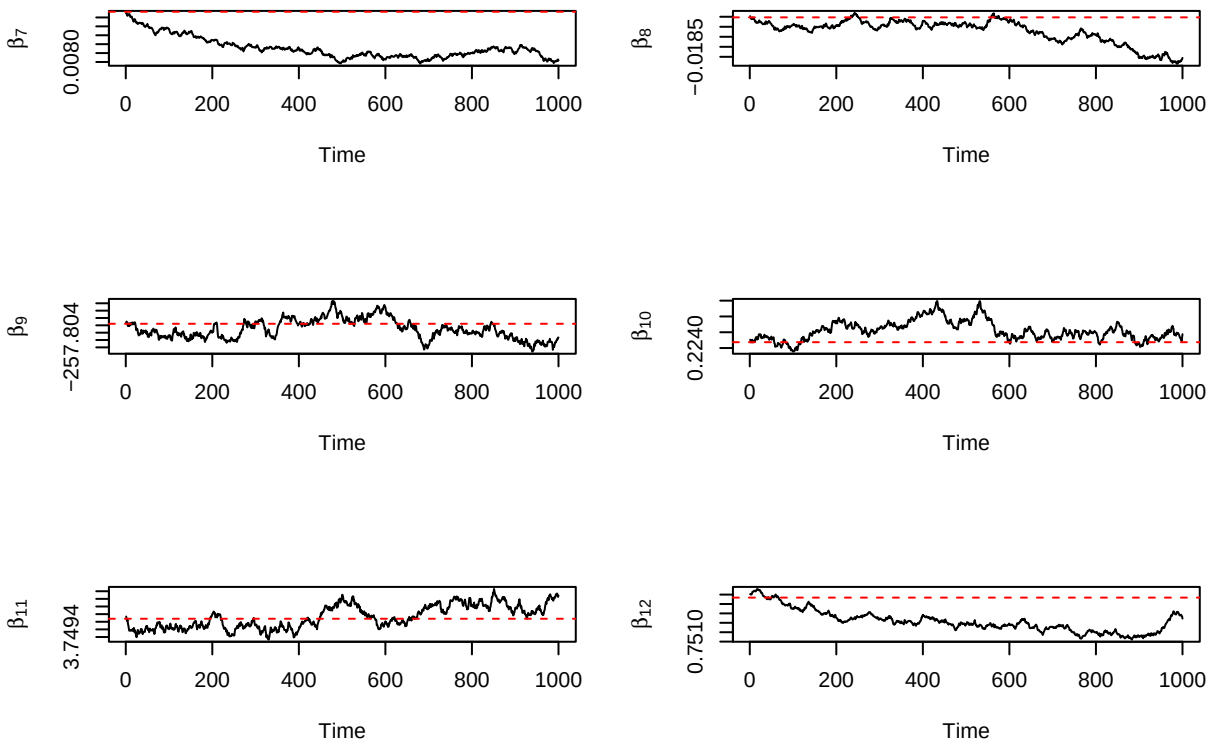
```
## [1] 0
```

This is very low, so we can adjust the scale parameter to find an acceptance rate around 20%.

```
#out <- metrop(out, scale = 0.005)  # 0.0075
out <- metrop(out, scale = c(0.005, 0.00005, 0.00005, 0.00005, 0.00005, 0.00005, 0.00005, 0.00005, 0.000
out$accept
```

```
## [1] 0.93
```


β0
242.4
0
200
400
600
800
1000
Time
β1
0.2730
0
200
400
600
800
1000
Time
β2
−2.5820
0
200
400
600
800
1000
Time
β3
0.5670
0
200
400
600
800
1000
Time
β4
0.2375
0
200
400
600
800
1000
Time
β5
−8.8170
0
200
400
600
800
1000
Time



VISUAL COMPARISON

β_0 (Intercept: 242.7625): ## β_1 (fixed_acidity: 0.275): ## β_2 (volatile_acidity: -2.581): ## β_3 (citric_acid: 0.568): ## β_4 (residual_sugar: 0.239): ## β_5 (chlorides: -8.816): ## β_6 (free_sulfur_dioxide: 0.011): β_7 (total_sulfur_dioxide: -0.017): ## β_8 (density: -257.798): ## β_9 (pH: 0.224): ## β_{10} (sulphates: 3.750): ## β_{11} (alcohol: 0.753): ##

```
summ <- matrix(NA, nrow=12, ncol=6)
for (i in 1:12) {
  summ[i,1] = mean(beta_mat_init1[2000:10000,i])
  summ[i,2] = sd(beta_mat_init1[2000:10000,i])
  summ[i,3] = mean(beta_mat2[12000:20000,i])
  summ[i,4] = sd(beta_mat2[12000:20000,i])
  summ[i,5] = mean(ts(out$batch)[,i])
  summ[i,6] = sd(ts(out$batch)[,i])
}
colnames(summ) <- c("Mean_MH", "SD_MH", "Mean_MH2", "SD_MH2", "Mean_Metrop", "SD_Metrop")
rownames(summ) <- c("Intercept", "fixed_acidity", "volatile_acidity", "citric_acid", "residual_sugar",
                    "chlorides", "free_sulfur_dioxide", "total_sulfur_dioxide", "density",
                    "pH", "sulphates", "alcohol")
```

```
knitr::kable(summ)
```

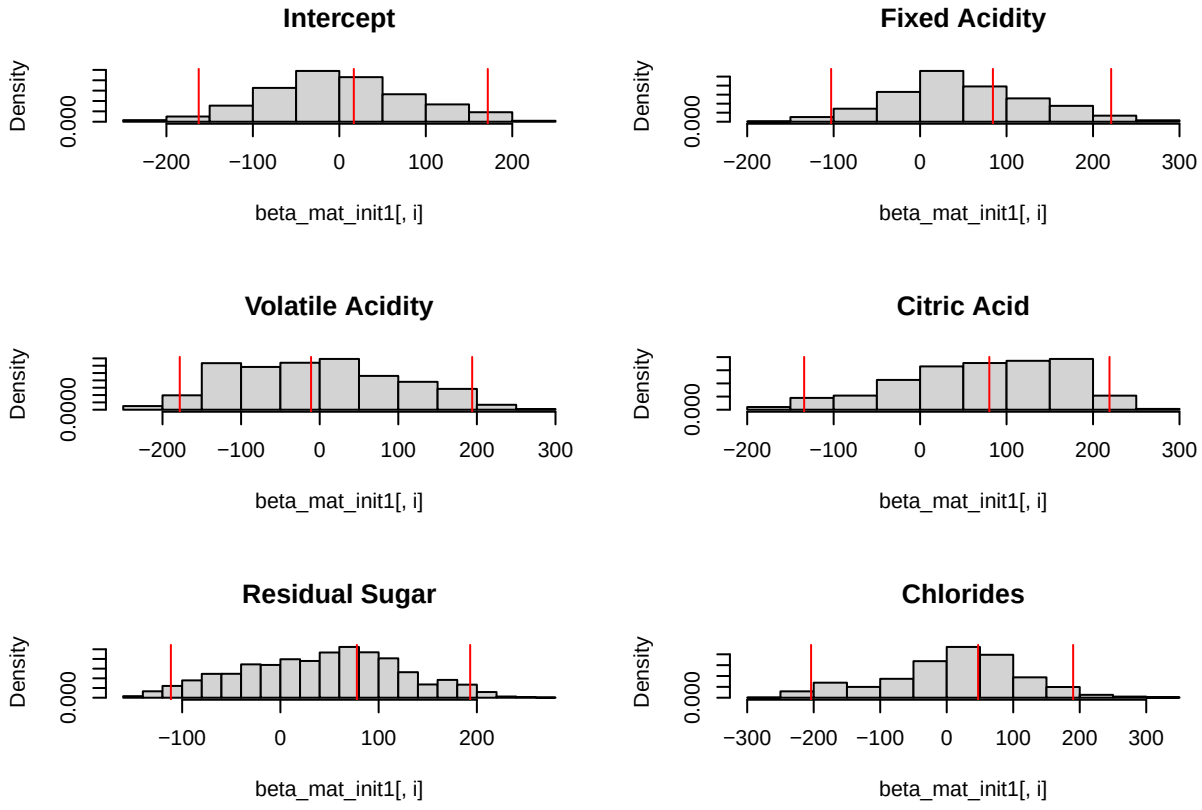
	Mean_MH	SD_MH	Mean_MH2	SD_MH2	Mean_Metrop	SD_Metrop
Intercept	16.745940	81.70296	-23.8801338	67.2089302	242.5185398	0.1163592

	Mean_MH	SD_MH	Mean_MH2	SD_MH2	Mean_Metrop	SD_Metrop
fixed_acidity	84.189866	63.36457	1.8821541	0.5150979	0.2744363	0.0006261
volatile_acidity	-10.962062	112.16421	-13.5271195	7.4595906	-2.5813830	0.0004172
citric_acid	80.060129	88.88544	-21.9467732	10.4131110	0.5680549	0.0003787
residual_sugar	77.855458	61.04691	-0.5729557	0.9899764	0.2385122	0.0004638
chlorides	47.211830	89.22684	-11.1725537	19.9044993	-8.8158180	0.0006994
free_sulfur_dioxide	73.072953	72.11203	0.0553317	0.0801461	0.0087703	0.0005885
total_sulfur_dioxide	66.625600	50.65816	0.0579425	0.0180559	-0.0172332	0.0005932
density	-32.329112	76.01111	-15.7121389	69.1071824	-257.7990758	0.0026897
pH	-4.095097	107.15831	7.5764954	5.0847301	0.2245523	0.0003027
sulphates	22.811505	68.55784	-0.7342002	2.9623051	3.7499248	0.0003031
alcohol	33.885668	52.56204	-0.3407030	0.5266895	0.7520774	0.0005853

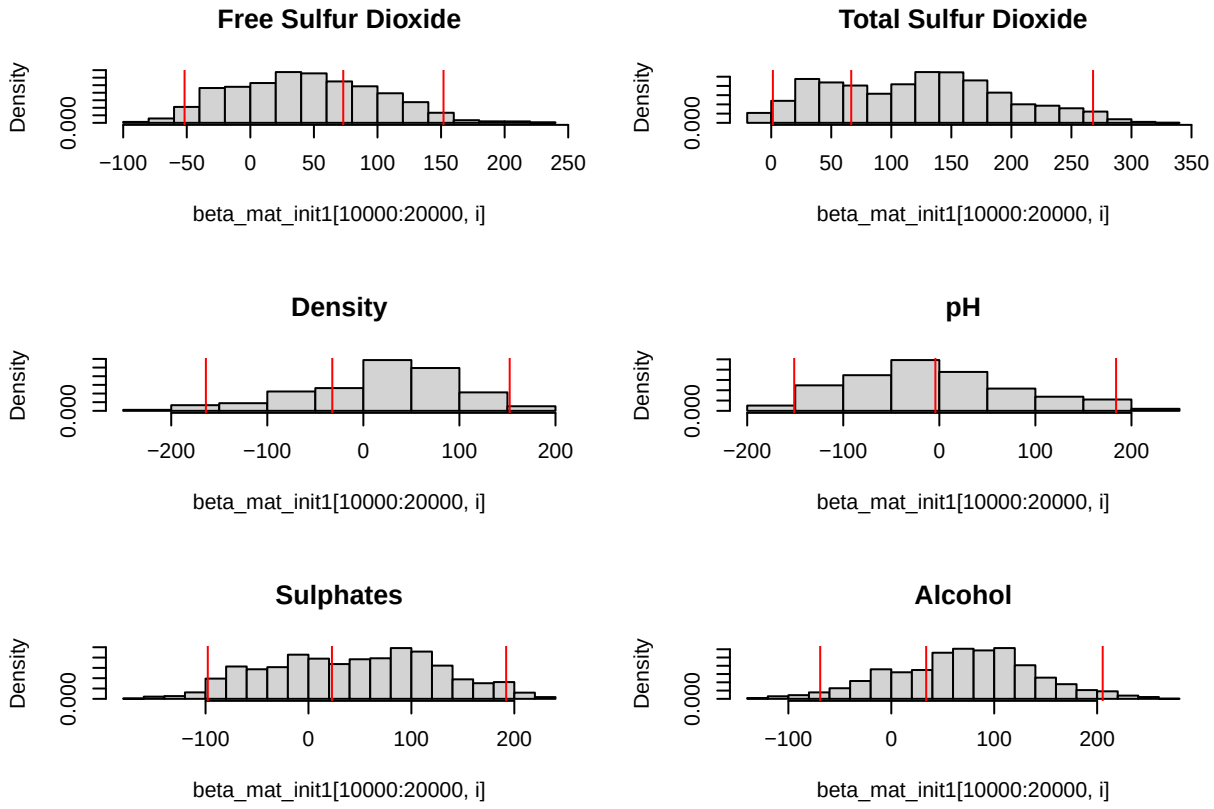
Histogram of spread of chain values

```
# hist(beta_mat_init1[,1])
# ci0 <- quantile(beta_mat_init1[,1], prob=c(.025,.975))
coeffs <- c("Intercept", "Fixed Acidity", "Volatile Acidity", "Citric Acid", "Residual Sugar",
                    "Chlorides", "Free Sulfur Dioxide", "Total Sulfur Dioxide", "Density",
                    "pH", "Sulphates", "Alcohol")
```

```
par(mfrow=c(3,2))
for (i in 1:6) {
hist(beta_mat_init1[,i], main=coeffs[i], prob=TRUE); abline(v=c(summ[i,1],quantile(beta_mat_init1[,i], p
}
```



```
par(mfrow=c(3,2))
for (i in 7:12) {
hist(beta_mat_init1[10000:20000,i], main=coeffs[i], prob=TRUE); abline(v=c(summ[i,1],quantile(beta_mat_
}
```



EXTRA

The following is not part of the scope of the analysis. It contains normalised data as a comparison to the un-normalised. The outcome is that each coefficient chain is closer to convergence to the MLE value. The second model using `Omega_prop` <- solve(t(X) %*% X) is a much more stable chain compared to the same using un-normalised data.

Normalise Data

Normalised data to compare results

```
response <- wine$good
wine = wine[, -c(12, 13, 14)]

normalise <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}
```

```
wine_norm <- as.data.frame(lapply(wine, normalise))
y = response
x = wine_norm
# Combine data

dat <- cbind(x, y)
names(dat)[12] <- "GOOD"
# test <- dat[1:20,]  # match with testPred
# train <- dat[c(21:100),]
y = dat$GOOD
# x = train[,c(1:11)]
X = cbind(rep(1, nrow(dat)), dat$fixed_acidity, dat$volatile_acidity, dat$citric_acid,
          dat$residual_sugar, dat$chlorides, dat$free_sulfur_dioxide, dat$total_sulfur_dioxide,
          dat$density, dat$pH, dat$sulphates, dat$alcohol)
```

Fit using normalised covariates

```
library(Metrics)
#
# fit <- glm(GOOD ~ ., data = dat)  #; fit$coefficients
# pred <- predict(fit, test[, c(-12)])
# rmse(actual = test$GOOD, predicted=pred)
# summary(fit)

fit <- glm(GOOD ~ ., data = dat, family = binomial(link="logit"))  #; fit$coefficients
pred <- predict(fit, dat[, c(-12)])
rmse(actual = dat$GOOD, predicted=pred)
```

```
## [1] 3.370479
```

```
lpost.LR <- function(beta, x, y) {
  eta <- as.numeric(x %*% beta)
  logp <- eta - log(1 + exp(eta))
  logq <- log(1 - exp(logp))
  logl <- sum(logp[y==1]) + sum(logq[y==0])
  lprior <- sum(dnorm(beta, 0, 100,log = T))
  return(logl + lprior)
}
```

```
S <- 10000
init <- matrix(data=c(runif(48, min = 0, max = 0)), nrow=4, ncol=12, byrow = T)

# First initialisation
beta_mat_init1 <- matrix(NA, nrow = S, ncol = ncol(X))
k <- ncol(beta_mat_init1)
beta_mat_init1[1,] <- init[1,]
#beta_mat_init1[1,] <- as.numeric(coefficients(fit))  # initialise with MLE of each coefficient
acc <- 0
for (iter in 2:S) {
  # simulate all (k) values using previous value of beta as mean and set sd
  beta_star <- rnorm(k, beta_mat_init1[iter-1,], 0.1)
  # compute target distribution for proposed value
```

```
  newpost = lpost.LR(beta_star, X, y)
  # compute target distribution for old value
  oldpost = lpost.LR(beta_mat_init1[iter-1,], X, y)  # symmetric dist => no ratio computed

  # acceptance step, in log scale
  if (runif(1,0,1) > exp(newpost - oldpost)) {
    # chain doesn't move
    beta_mat_init1[iter,] = beta_mat_init1[iter-1,]
  } else {
    # add to chain and add 1 to counter
    beta_mat_init1[iter,] = beta_star
    acc=acc + 1
  }
  #if (iter%%1000 == 0) {print(c(iter, acc/iter))}
}
print(c(iter, acc/iter))
```

```
## [1] 1.000e+04 5.095e-01
```

```
# Second initialisation
beta_mat_init2 <- matrix(NA, nrow = S, ncol = ncol(X))
k <- ncol(beta_mat_init2)
beta_mat_init2[1,] <- init[2,]
acc <- 0
for(iter in 2:S){
  beta_star <- rnorm(k, beta_mat_init2[iter-1,], 0.1)
  newpost = lpost.LR(beta_star, X, y)
  oldpost = lpost.LR(beta_mat_init2[iter-1,], X, y)
  if (runif(1,0,1) > exp(newpost - oldpost)) {
    beta_mat_init2[iter,] = beta_mat_init2[iter-1,]
  } else {
    beta_mat_init2[iter,] = beta_star
    acc=acc + 1
  }
  #if (iter%%1000==0) {print(c(iter,acc/iter))}
}
print(c(iter, acc/iter))
```

```
## [1] 1.000e+04 5.077e-01
```

```
# Third initialisation
beta_mat_init3 <- matrix(NA,nrow=S,ncol=ncol(X))
k <- ncol(beta_mat_init3)
beta_mat_init3[1,] <- init[3,]
acc <- 0
for(iter in 2:S){
  beta_star <- rnorm(k,beta_mat_init3[iter-1,], 0.1)
  newpost=lpost.LR(beta_star,X,y)
  oldpost=lpost.LR(beta_mat_init3[iter-1,],X,y)
  if(runif(1,0,1)>exp(newpost-oldpost)){
    beta_mat_init3[iter,]=beta_mat_init3[iter-1,]
  } else{
```

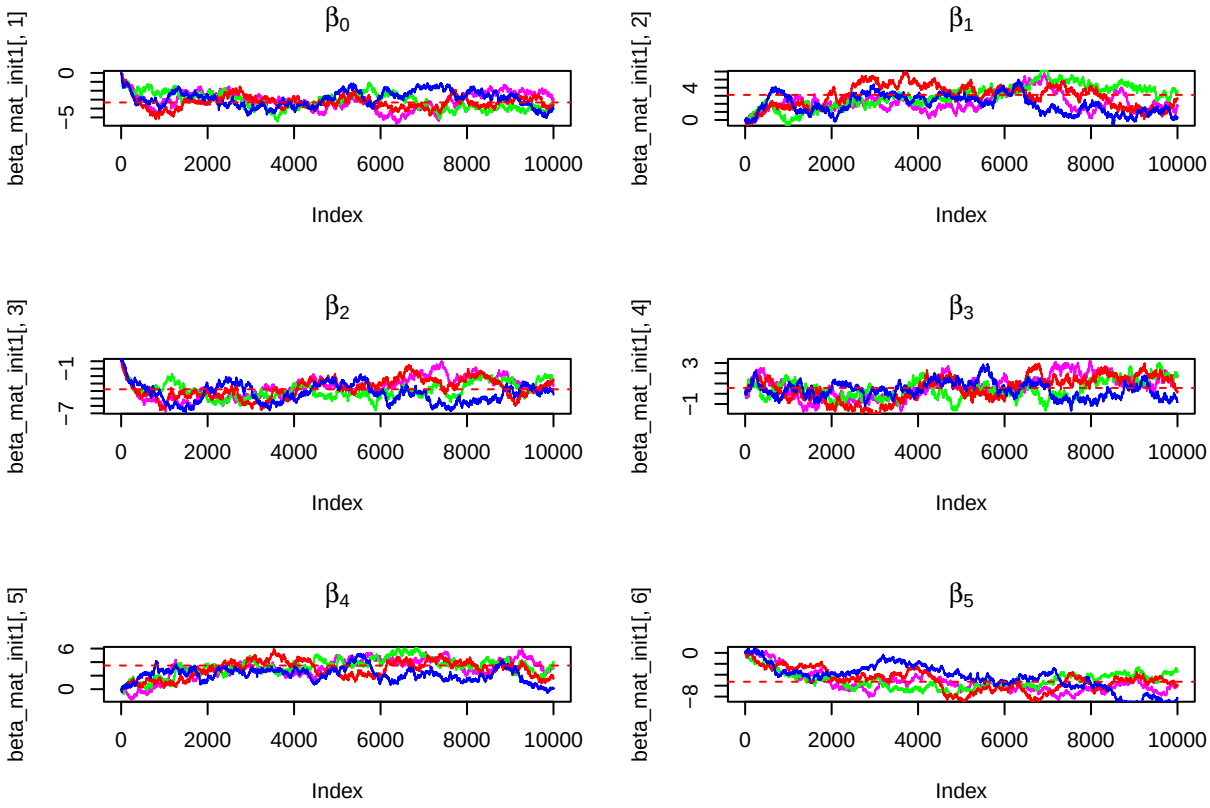
```
    beta_mat_init3[iter,]=beta_star
    acc=acc+1
  }
  #if(iter%%1000==0){print(c(iter,acc/iter))}
}
print(c(iter, acc/iter))
```

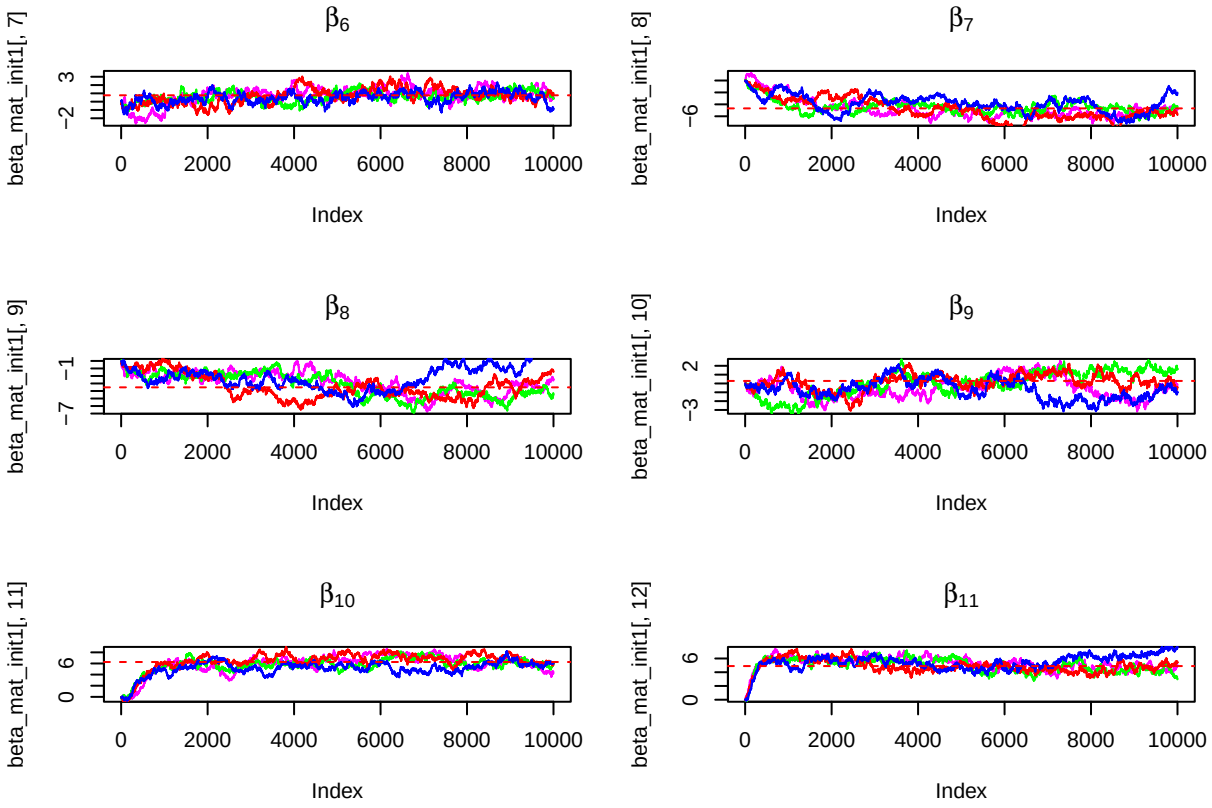
```
## [1] 1.00e+04 5.06e-01
```

```
# Forth initialisation
beta_mat_init4 <- matrix(NA,nrow=S,ncol=ncol(X))
k <- ncol(beta_mat_init4)
beta_mat_init4[1,] <- init[4,]
acc <- 0
for(iter in 2:S){
  beta_star <- rnorm(k,beta_mat_init4[iter-1,], 0.1)
  newpost=lpost.LR(beta_star,X,y)
  oldpost=lpost.LR(beta_mat_init4[iter-1,],X,y)
  if(runif(1,0,1)>exp(newpost-oldpost)){
    beta_mat_init4[iter,]=beta_mat_init4[iter-1,]
  } else{
    beta_mat_init4[iter,]=beta_star
    acc=acc+1
  }
  #if(iter%%1000==0){print(c(iter,acc/iter))}
}
print(c(iter, acc/iter))
```

```
## [1] 1.000e+04 5.152e-01
```

Plot the chains for each coefficient (same plot)


β0
β1
β2
β3
β4
β5
beta_mat_init1[, 1]
beta_mat_init1[, 2]
beta_mat_init1[, 3]
beta_mat_init1[, 4]
beta_mat_init1[, 5]
beta_mat_init1[, 6]
Index
0
2000
4000
6000
8000
10000

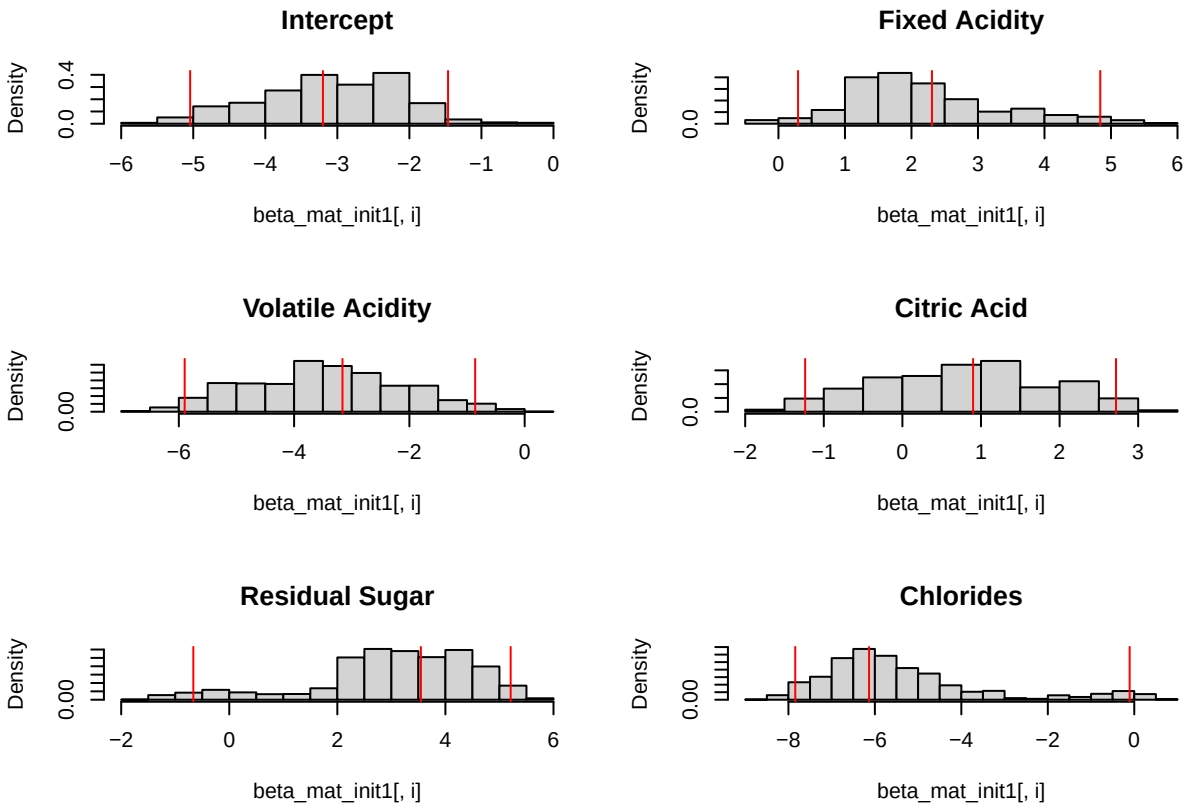



```
summ <- matrix(NA, nrow=12, ncol=2)
for (i in 1:12) {
  summ[i,1] = mean(beta_mat_init1[2000:10000,i])
  summ[i,2] = sd(beta_mat_init1[2000:10000,i])
}
colnames(summ) <- c("Mean_MH", "SD_MH")
rownames(summ) <- c("Intercept", "fixed_acidity", "volatile_acidity", "citric_acid", "residual_sugar",
                    "chlorides", "free_sulfur_dioxide", "total_sulfur_dioxide", "density",
                    "pH", "sulphates", "alcohol")
```

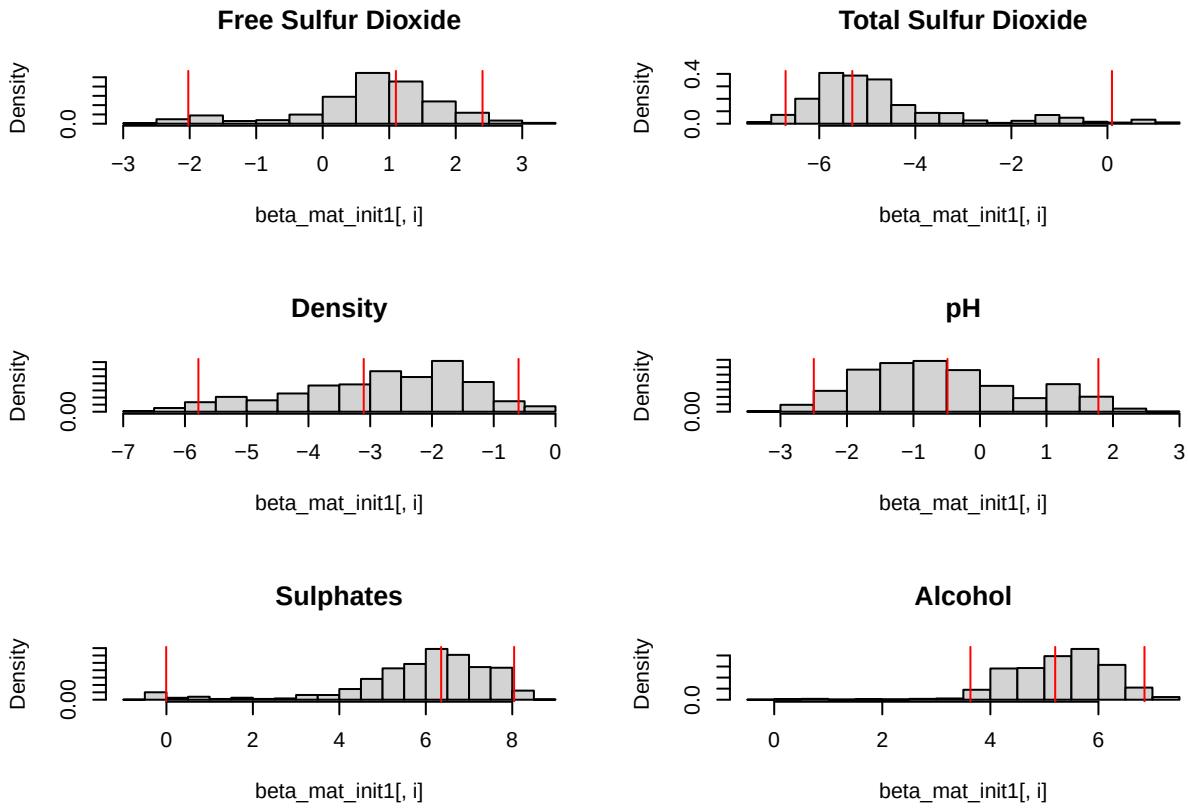
Histogram of spread of chain values

```
# hist(beta_mat_init1[,1])
# ci0 <- quantile(beta_mat_init1[,1], prob=c(.025,.975))
coeffs <- c("Intercept", "Fixed Acidity", "Volatile Acidity", "Citric Acid", "Residual Sugar",
                    "Chlorides", "Free Sulfur Dioxide", "Total Sulfur Dioxide", "Density",
                    "pH", "Sulphates", "Alcohol")
```

```
par(mfrow=c(3,2))
for (i in 1:6) {
hist(beta_mat_init1[,i], main=coeffs[i], prob=TRUE); abline(v=c(summ[i,1],quantile(beta_mat_init1[,i],
}
```




```
par(mfrow=c(3,2))
for (i in 7:12) {
hist(beta_mat_init1[,i], main=coeffs[i], prob=TRUE); abline(v=c(summ[i,1],quantile(beta_mat_init1[,i],
}
```

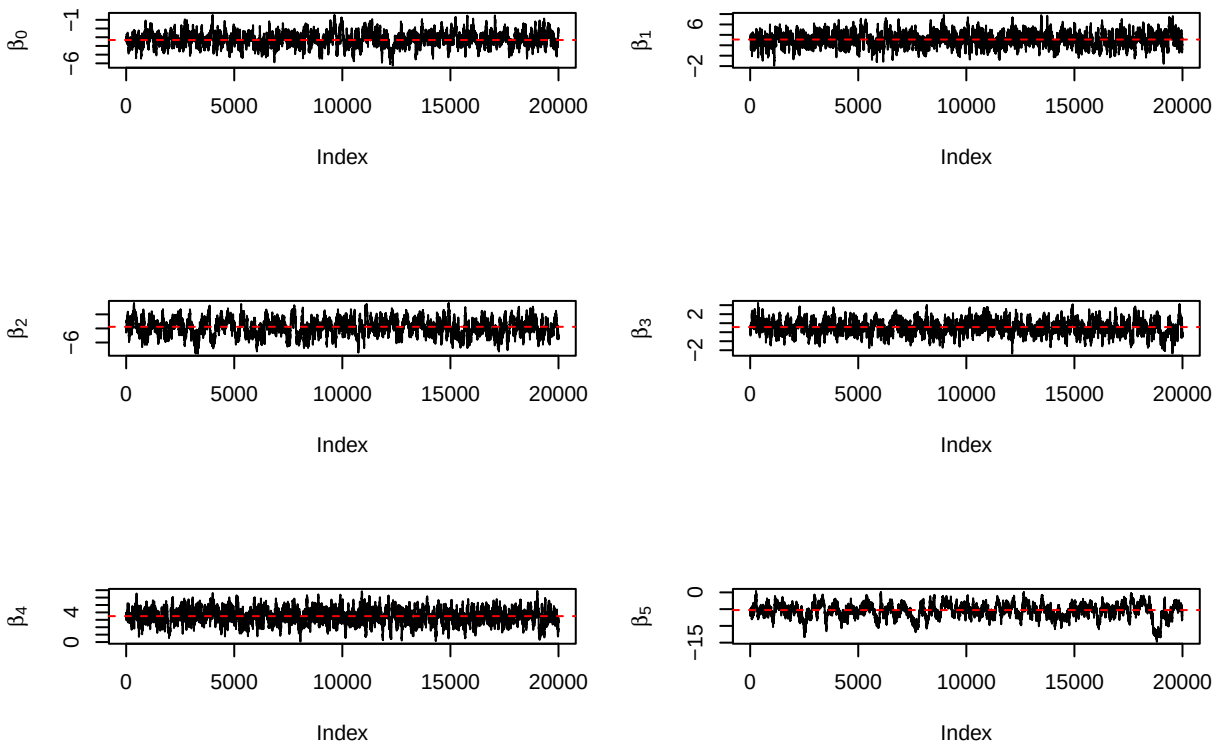


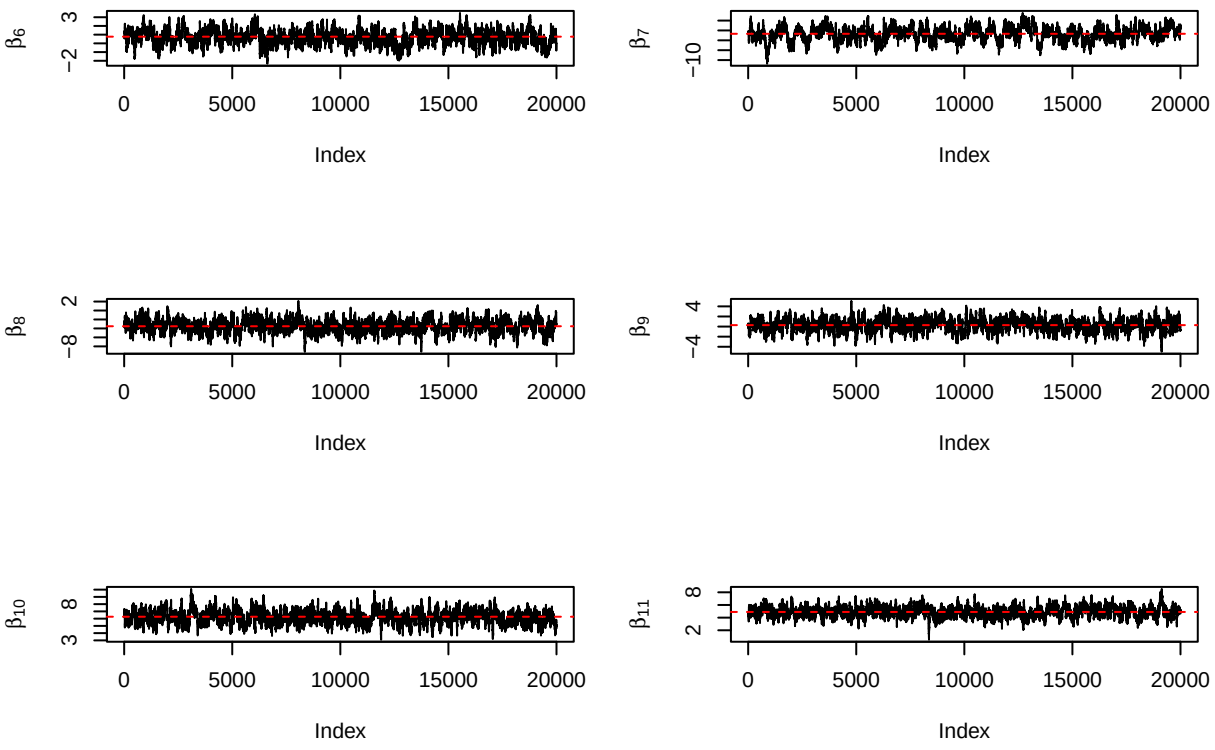

```
S <- 20000
beta_mat2 <- matrix(NA, nrow = S, ncol = ncol(X))
beta_mat2[1,] <- as.numeric(coefficients(fit))

y_new <- c(1)
x_new <- c(1, fixed_acidity, volatile_acidity, citric_acid, residual_sugar,
           chlorides, free_sulfur_dioxide, total_sulfur_dioxide, density,
           pH, sulphates, alcohol)
```

```
library(mvtnorm)

# prediction

Omega_prop <- solve(t(X) %*% X)
k <- ncol(beta_mat2)
acc <- 0
for(iter in 2:S)
{
  # 1. Propose a new set of values
  beta_star <- rmvnorm(1, beta_mat2[iter-1,], 1.5 * Omega_prop)

  # 2. Compute the posterior density on the proposed value and on the old value
  newpost=lpost.LR(t(beta_star), X, y)
  oldpost=lpost.LR(matrix(beta_mat2[iter-1,], ncol=1), X, y)

  # 3. Acceptance step
```

```
  if (runif(1, 0, 1) > exp(newpost - oldpost)) {
    beta_mat2[iter,] = beta_mat2[iter-1,]
  } else {
    beta_mat2[iter,] = beta_star
    acc = acc + 1
  }
  # 4. Print the stage of the chain
  if (iter%%1000 == 0){ print(c(iter, acc/iter)) }

  # 5. Prediction
  p_new <- exp(sum(beta_mat2[iter,] * x_new) ) / (1 + exp(sum(beta_mat2[iter,] * x_new) ))
  y_new[iter] <- rbinom(1,1,prob=p_new)
}
```

```
## [1] 1000.000    0.565
## [1] 2000.000    0.557
## [1] 3000.000    0.555
## [1] 4000.00000    0.54425
## [1] 5000.0000    0.5478
## [1] 6000.0000    0.5435
## [1] 7000.0000000    0.5437143
## [1] 8000.0000    0.5445
## [1] 9000.000    0.548
## [1] 1.000e+04 5.469e-01
## [1] 1.100000e+04 5.480909e-01
## [1] 1.200000e+04 5.469167e-01
## [1] 1.300000e+04 5.480769e-01
## [1] 1.40e+04 5.47e-01
## [1] 1.500000e+04 5.464667e-01
## [1] 1.600000e+04 5.456875e-01
## [1] 1.700000e+04 5.452941e-01
## [1] 1.800000e+04 5.439444e-01
## [1] 1.900000e+04 5.438947e-01
## [1] 2.000e+04 5.445e-01
```


β0
−6
−1
0
5000
10000
15000
20000
Index
β1
−2
6
0
5000
10000
15000
20000
Index
β2
−6
0
5000
10000
15000
20000
Index
β3
−2
2
0
5000
10000
15000
20000
Index
β4
0
4
0
5000
10000
15000
20000
Index
β5
−15
0
0
5000
10000
15000
20000
Index


β6
−2
3
0
5000
10000
15000
20000
Index
β7
−10
0
5000
10000
15000
20000
Index
β8
−8
2
0
5000
10000
15000
20000
Index
β9
−4
4
0
5000
10000
15000
20000
Index
β10
3
8
0
5000
10000
15000
20000
Index
β11
2
8
0
5000
10000
15000
20000
Index