SMDS-2-Final Project

Theodoros Sofianos-1867968

July 7, 2020

Diabetes dataset: an in-depth fully Bayesian analysis.

In this project, we will perform our analysis on the diabetes dataset(https://github.com/MateLabs/Public-Datasets/blob/master/Datasets/diabetes.csv). The dataset consists of 8 independent numerical variables and 1 binary response variable,whether the patient actually has diabetes or not.

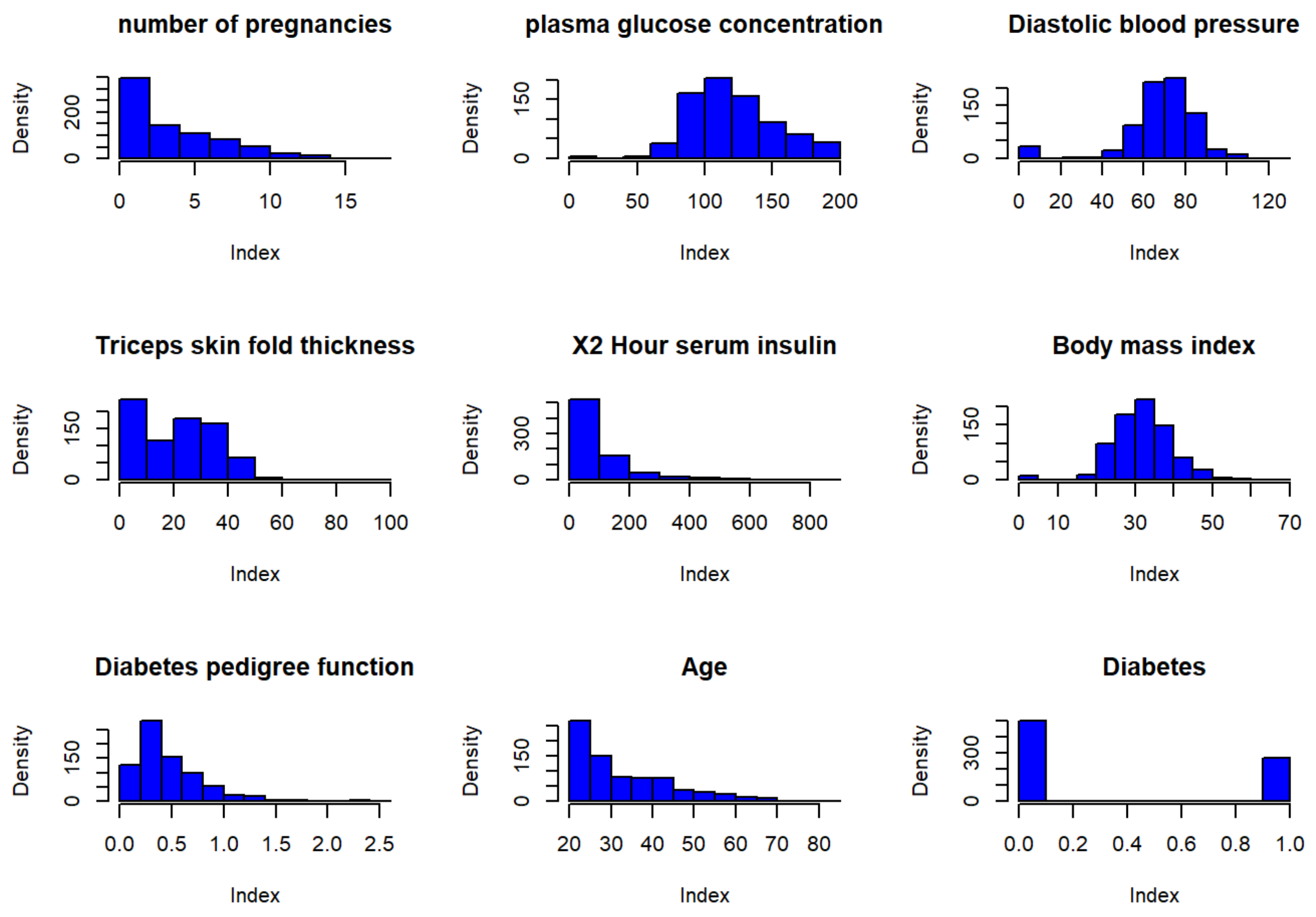
```
require(corrplot)


data=read.csv('D:\\sapienza\\SDS2\\diabetes.csv')

Y=ifelse(data$Class == "positive",1,0)

data$Class<-NULL

n<-length(Y)
#data$Class<-Y


sum(is.na(data))
```

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

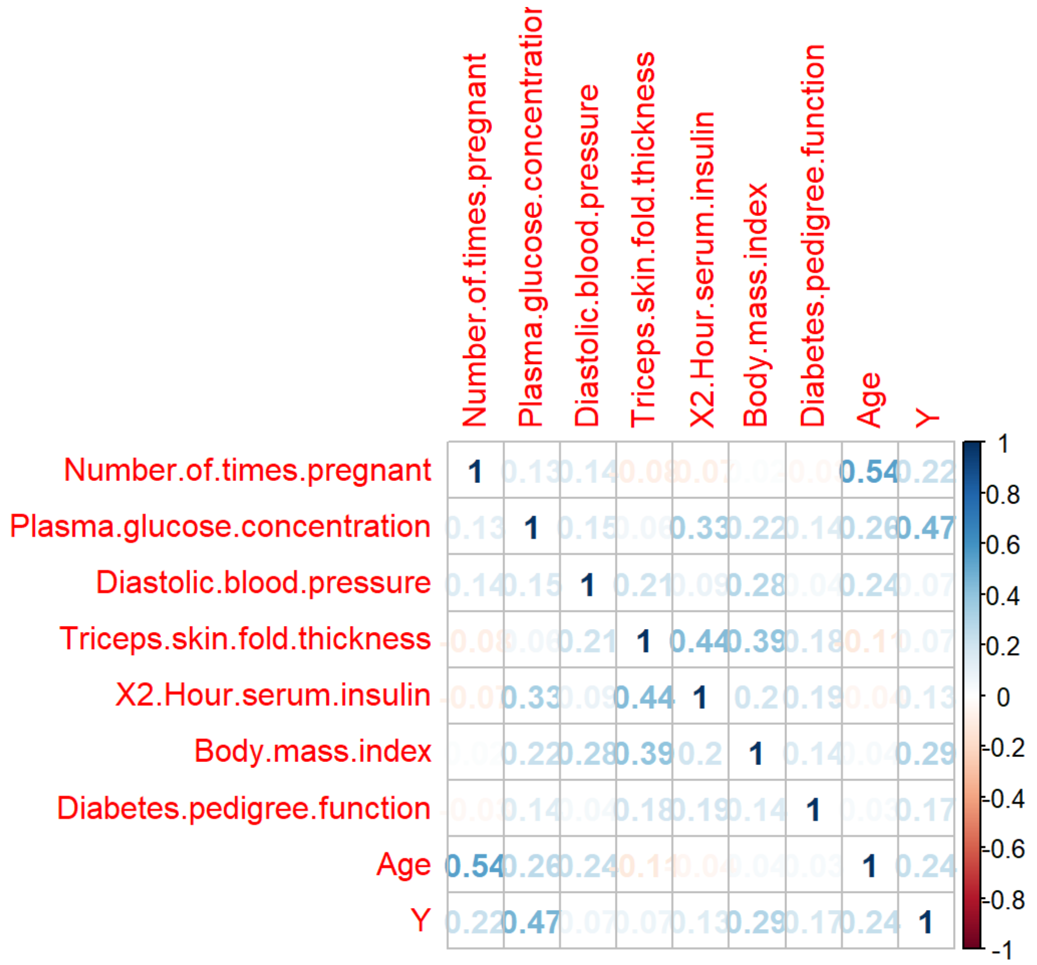
```
summary(data)
```

```
##  Number.of.times.pregnant Plasma.glucose.concentration
##  Min.   : 0.000           Min.   :  0.0
##  1st Qu.: 1.000           1st Qu.: 99.0
##  Median : 3.000           Median :117.0
##  Mean   : 3.845           Mean   :120.9
##  3rd Qu.: 6.000           3rd Qu.:140.2
##  Max.   :17.000           Max.   :199.0
##  Diastolic.blood.pressure Triceps.skin.fold.thickness
##  Min.   :  0.00           Min.   : 0.00
##  1st Qu.: 62.00           1st Qu.: 0.00
##  Median : 72.00           Median :23.00
##  Mean   : 69.11           Mean   :20.54
##  3rd Qu.: 80.00           3rd Qu.:32.00
##  Max.   :122.00           Max.   :99.00
##  X2.Hour.serum.insulin Body.mass.index Diabetes.pedigree.function
##  Min.   :  0.0         Min.   : 0.00   Min.   :0.0780
##  1st Qu.:  0.0         1st Qu.:27.30   1st Qu.:0.2437
##  Median : 30.5         Median :32.00   Median :0.3725
##  Mean   : 79.8         Mean   :31.99   Mean   :0.4719
##  3rd Qu.:127.2         3rd Qu.:36.60   3rd Qu.:0.6262
##  Max.   :846.0         Max.   :67.10   Max.   :2.4200
##       Age
##  Min.   :21.00
##  1st Qu.:24.00
##  Median :29.00
##  Mean   :33.24
##  3rd Qu.:41.00
##  Max.   :81.00
```

```
par(mfrow=c(3,3))
hist(main='number of pregnancies',data$Number.of.times.pregnant,xlab='Index',ylab='Density',col='blue')
hist(main='plasma glucose concentration',data$Plasma.glucose.concentration,xlab='Index',ylab='Density',col='blue'
)
hist(main='Diastolic blood pressure',data$Diastolic.blood.pressure,xlab='Index',ylab='Density',col='blue')
hist(main='Triceps skin fold thickness',data$Triceps.skin.fold.thickness,xlab='Index',ylab='Density',col='blue')
hist(main='X2 Hour serum insulin',data$X2.Hour.serum.insulin,xlab='Index',ylab='Density',col='blue')
hist(main='Body mass index',data$Body.mass.index,xlab='Index',ylab='Density',col='blue')
hist(main='Diabetes pedigree function',data$Diabetes.pedigree.function,xlab='Index',ylab='Density',col='blue')
hist(main='Age',data$Age,xlab='Index',ylab='Density',col='blue')
hist(main='Diabetes',Y,xlab='Index',ylab='Density',col='blue')
```



```
par(mfrow=c(1,1))
corrplot(cor(cbind(data,Y)),method = 'number')
```



Above, we can see the histograms of the independent variables and the correlations between them.We can observe that the highest correlation between variables is for the couple of Age-Number of pregnancies,something that intuitively makes sense.We can also see that the highest correlation for our response variable is with the variable Plasma glucose concentration.Furthermore,we can see that some of our independent variables have almost zero correlation with it,so it seems that we do not need all of the variables for our model.Since some of the independent variables of our dataset are hard to grasp without some medicine background and in order to deal with multicollinearity issues as well,we will perform a Principal Component Analysis of the dataset.

PCA

```
require(dplyr)
pca=prcomp(data,center = T,scale. = T)
```

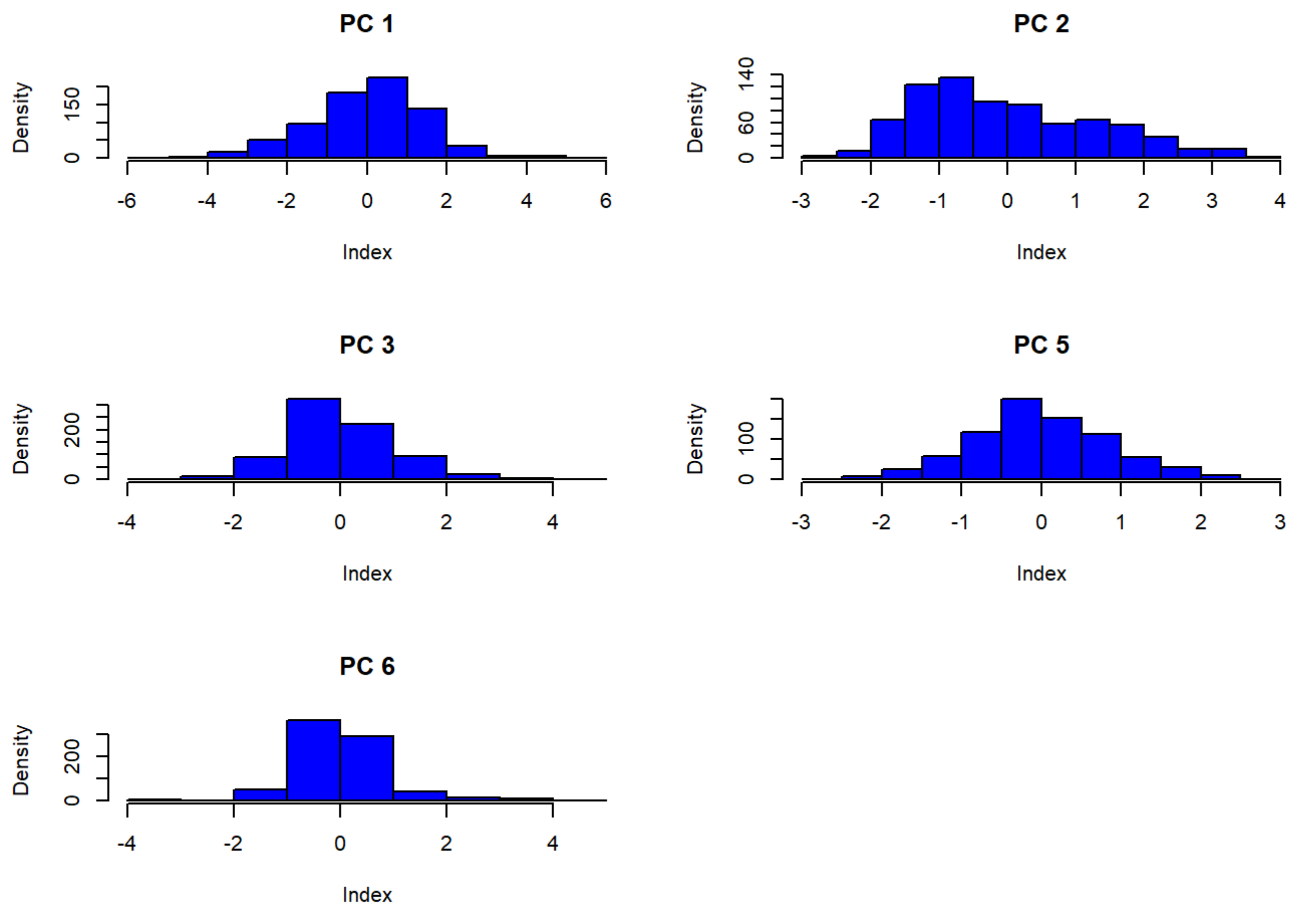
```
X= pca$x



summary(pca)
```

```
## Importance of components:
##                           PC1    PC2    PC3    PC4     PC5     PC6     PC7
## Standard deviation     1.4472 1.3158 1.0147 0.9357 0.87312 0.82621 0.64793
## Proportion of Variance 0.2618 0.2164 0.1287 0.1094 0.09529 0.08533 0.05248
## Cumulative Proportion  0.2618 0.4782 0.6069 0.7163 0.81164 0.89697 0.94944
##                            PC8
## Standard deviation     0.63597
## Proportion of Variance 0.05056
## Cumulative Proportion  1.00000
```

```
lr=glm(Y~as.matrix(X),family = 'binomial')

summary(lr)
```

```
##
## Call:
## glm(formula = Y ~ as.matrix(X), family = "binomial")
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5566  -0.7274  -0.4159   0.7267   2.9297
##
## Coefficients:
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -0.87110    0.09694  -8.986  < 2e-16 ***
## as.matrix(X)PC1 -0.78619    0.07649 -10.278  < 2e-16 ***
## as.matrix(X)PC2  0.42485    0.06901   6.157 7.43e-10 ***
## as.matrix(X)PC3  0.50276    0.09611   5.231 1.68e-07 ***
## as.matrix(X)PC4 -0.07559    0.09699  -0.779   0.4358
## as.matrix(X)PC5  0.48278    0.10629   4.542 5.57e-06 ***
## as.matrix(X)PC6  0.83868    0.12234   6.855 7.13e-12 ***
## as.matrix(X)PC7  0.16912    0.13547   1.248   0.2119
## as.matrix(X)PC8  0.35795    0.14511   2.467   0.0136 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 993.48  on 767  degrees of freedom
## Residual deviance: 723.45  on 759  degrees of freedom
## AIC: 741.45
##
## Number of Fisher Scoring iterations: 5
```

```
X=select(as.data.frame(X),PC1,PC2,PC3,PC5,PC6)

par(mfrow=c(3,2))
hist(main='PC 1',X[,1],xlab='Index',ylab='Density',col='blue')
hist(main='PC 2',X[,2],xlab='Index',ylab='Density',col='blue')
hist(main='PC 3',X[,3],xlab='Index',ylab='Density',col='blue')
hist(main='PC 5',X[,4],xlab='Index',ylab='Density',col='blue')
hist(main='PC 6',X[,5],xlab='Index',ylab='Density',col='blue')
```



First of all, we see the importance of the 8 Principal Components. The first 5 components can ‘explain’ more than 80% of the total variance of the dataset.However, when fitting a logistic regression to these components, the component number 4 is not statistically significant,in contrast with the components 5 and 6, although it accounts for more than 10% of the total variance.Also, the component number 8 is statistically significant,although it accounts for 5% of the variance of the dataset.Based on these artifacts,we have decided to implement a logit model,with variables the Principal Components 1,2,3,5,6. We can see the histograms of the variables of our model in the graphic above.

Logit Model

Frequentist approach:

Our response variable follows an i.i.d. bernoulli distribution,with probability p_i for each patient i.The probability p can be calculated as: $p(i) = w_1 + w_2 * X_1(i) + w_3 * X_2(i) + w_4 * X_3(i) + w_5 * X_4(i) + w_6 * X_5(i), where X_1(i)..X(5)(i)$ are the original data of patient i,projected along the Principal Components mentioned above. Since the output is binary, the equation above is wrapped in a Sigmoid,to output values between 0 and 1.In terms of matrices, we can rewrite the above equation as :$p = \sigma(X^T W)$ The likelihood function is:

$$Pr(p|W) = \prod_{i=1}^{n} p(i)^{Y(i)} (1 - p(i))^{1-Y(i)}$$

and can be written as log:

$$\ln(Pr(p|W)) = \sum_{i=1}^{n} Y(i) \ln\ p(i) + (1 - Y(i)) \ln\ (1 - p(i))$$

, which is essentially the (minus) cross entropy loss of the logistic regression.

```
#MLE OF LINEAR MODEL

require(pROC)


sigmoid=function(a){

  return (1/(1+exp(-a)))
}

mle_log=function(theta){ #alpha,beta,gamma,tau




  x=cbind(X0=1,X)

  tau=sigmoid(theta%*%t(x))
  sum(Y*log(tau)+(1-Y)*log(1-tau))




}
MLE<-optim(c(0.3,-0.10,0.05,0.05,-0.05,0.05),fn=mle_log,control=list(fnscale=-1),hessian = T) #MLE

mle_fit_values=MLE$par[1]+MLE$par[2]*X[,1]+MLE$par[3]*X[,2]+MLE$par[4]*X[,3]+MLE$par[5]*X[,4]+MLE$par[6]*X[,5]


lr=glm(Y~as.matrix(X),family = 'binomial')

cat('parameters via optim of MLE function:',MLE$par,'parameters via glm function:',lr$coefficients)
```

```
## parameters via optim of MLE function: -0.8676375 -0.7814026 0.4191899 0.4817931 0.4308103 0.8127787 parameters
via glm function: -0.8775487 -0.7890416 0.4181201 0.4830196 0.4831452 0.8406516
```

```
confint.default(lr)
```

```
##                       2.5 %     97.5 %
## (Intercept)     -1.0671956 -0.6879019
## as.matrix(X)PC1 -0.9389237 -0.6391595
## as.matrix(X)PC2  0.2847913  0.5514489
## as.matrix(X)PC3  0.2959994  0.6700397
## as.matrix(X)PC5  0.2759427  0.6903477
## as.matrix(X)PC6  0.6012031  1.0801001
```

```
par(mfrow=c(1,2))
par(pty='s')
roc(Y,lr$fitted.values,plot = T)
```

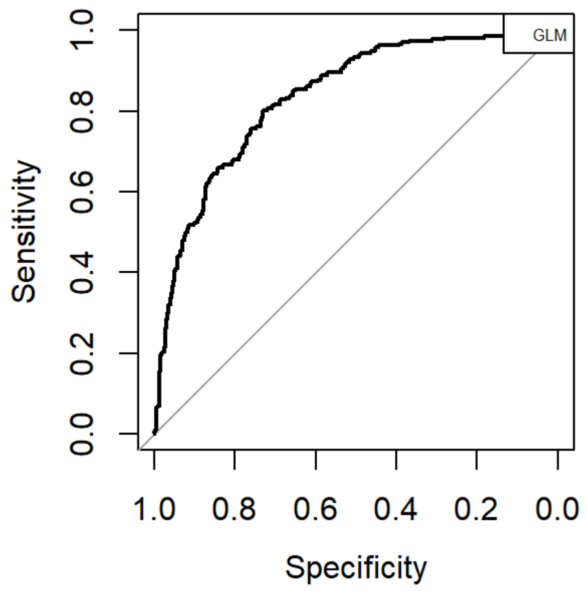
```
##
## Call:
```

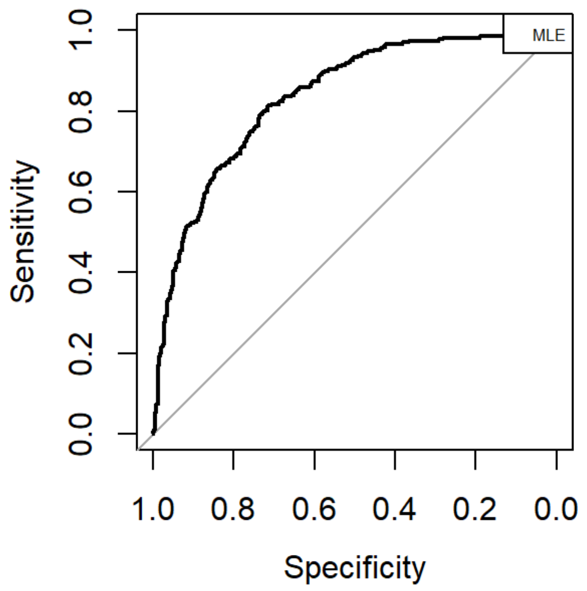
```
## roc.default(response = Y, predictor = lr$fitted.values, plot = T)
##
## Data: lr$fitted.values in 500 controls (Y 0) < 268 cases (Y 1).
## Area under the curve: 0.8364
```

```
legend('topright',legend = 'GLM',cex=0.5)
roc(Y,mle_fit_values,plot = T)
```

```
##
## Call:
## roc.default(response = Y, predictor = mle_fit_values, plot = T)
##
## Data: mle_fit_values in 500 controls (Y 0) < 268 cases (Y 1).
## Area under the curve: 0.8363
```

```
legend('topright',legend = 'MLE',cex=0.5)
```





We can see that both methods

produce simiar values for the parameters and almost identical ROC curves

MCMC

For our bayesian inference on the model described, we will use non-conjugate approach, so we have to resort to Jags in order to sample from the unknown posterion,using Markov Chains. We will also assume non informative priors. The code is below:

```
require(R2jags)
require(statip)
```

```
modelLogit <- function() {
  # Likelihood
  for(i in 1:n)
  {
    y[i] ~ dbern(p[i])

    logit(p[i]) <- theta[1] + theta[2]*X1[i] +theta[3]*X2[i]+theta[4]*X3[i]+theta[5]*X4[i]+theta[6]*X5[i]
  }

  # Priors
  theta[1] ~ dnorm(0, 1.0E-6)
  for (i in 1:n_param+1)
  {
    theta[i] ~ dnorm(0, 1.0E-6)

  }
}

# Preparing data for JAGS ---------------------------------------------
X1=X[,1]
X2=X[,2]
X3=X[,3]
X4=X[,4]
X5=X[,5]
y=Y
n=nrow(X)
n_param=ncol(X)

dat.jags <- list("X1", "X2", "X3", "X4", "X5", "y", "n","n_param")



# Parameters ---------------------------------------------------------------

modLogit.params="theta"

# Starting values
```

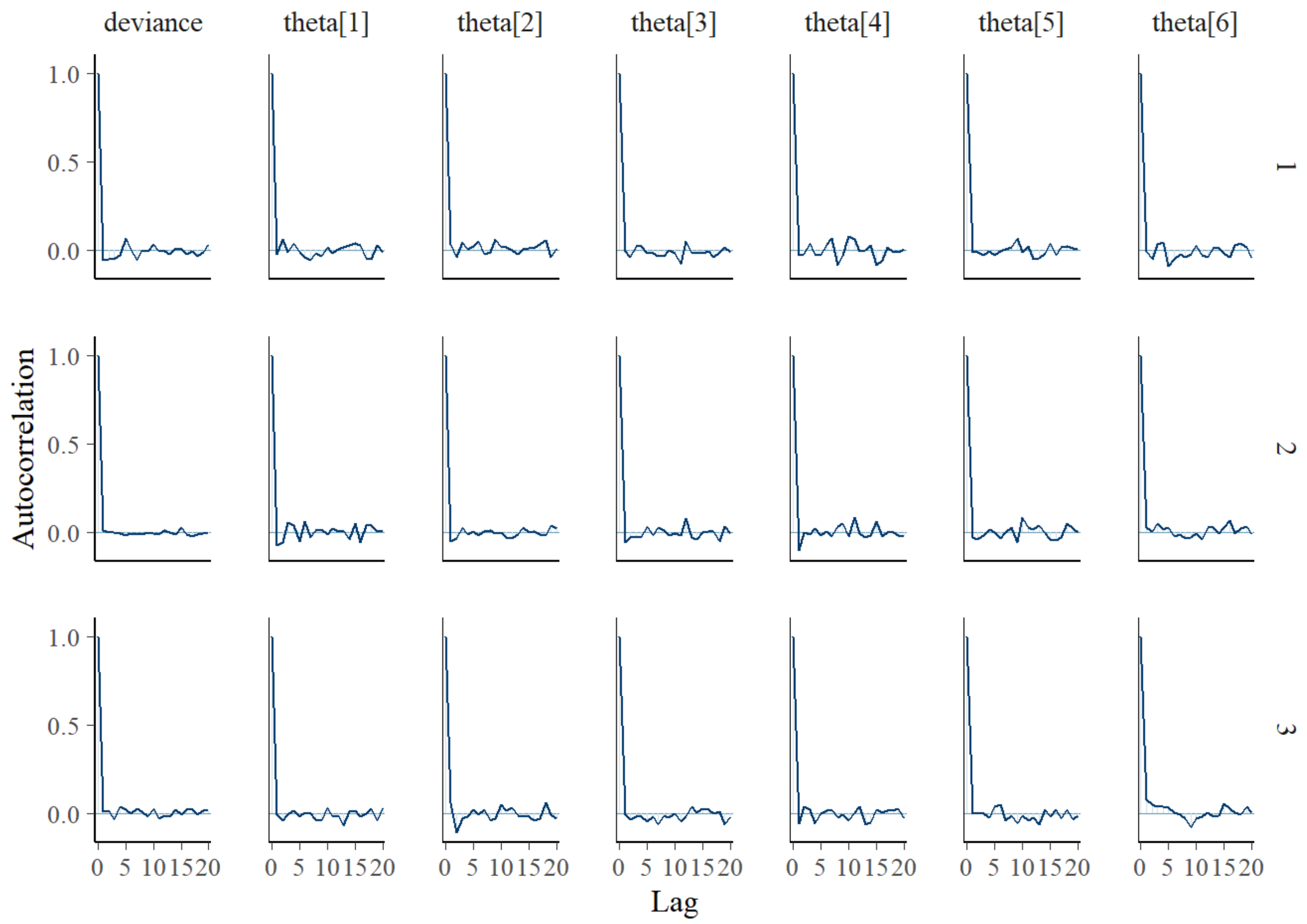
```
modLogit.inits <- function(){
  list("theta" = c(rnorm(1, 0, 1),rnorm(1, 0, 1),rnorm(1, 0, 1),rnorm(1, 0, 1),rnorm(1, 0, 1),rnorm(1, 0, 1))
       )
}



# Run JAGS ----------------------------------------------------------------

set.seed(123)
modLogit.fit <- jags(data = dat.jags,                                    # DATA
                      model.file = modelLogit, inits = modLogit.inits,          # MODEL
                      parameters.to.save = modLogit.params,
                      n.chains = 3, n.iter = 9000, n.burnin = 1000, n.thin=10)      # MCMC
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 768
##    Unobserved stochastic nodes: 6
##    Total graph size: 9994
##
## Initializing model
```

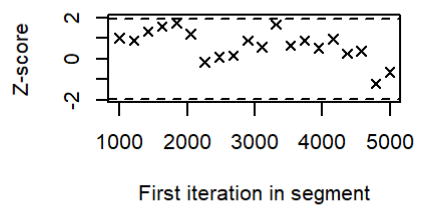
```
modLogit.fit
```

```
## Inference for Bugs model at "C:/Users/teoso/AppData/Local/Temp/RtmpCKRd8h/model31c82ec57dde.txt", fit using ja
gs,
##  3 chains, each with 9000 iterations (first 1000 discarded), n.thin = 10
##  n.sims = 2400 iterations saved
##          mu.vect sd.vect    2.5%     25%     50%     75%   97.5%  Rhat
## theta[1]  -0.882   0.097  -1.081  -0.945  -0.883  -0.817  -0.693 1.002
## theta[2]  -0.798   0.077  -0.953  -0.848  -0.797  -0.746  -0.655 1.001
## theta[3]   0.423   0.071   0.289   0.377   0.423   0.469   0.561 1.001
## theta[4]   0.491   0.094   0.311   0.426   0.488   0.556   0.676 1.002
## theta[5]   0.489   0.106   0.287   0.421   0.486   0.561   0.693 1.001
## theta[6]   0.852   0.126   0.613   0.768   0.850   0.932   1.105 1.002
## deviance 737.835   6.693 732.931 735.229 737.104 739.344 745.774 1.104
##          n.eff
## theta[1]  1100
## theta[2]  2400
## theta[3]  2400
## theta[4]  1800
## theta[5]  1800
## theta[6]  2400
## deviance  2400
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 22.4 and DIC = 760.3
## DIC is an estimate of expected predictive error (lower deviance is better).
```

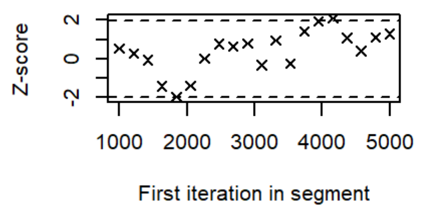
```
# We can observe that the point estimates of the parameters with the MCMC are close to the ones from the MLE.


chainArray <- modLogit.fit$BUGSoutput$sims.array

library(bayesplot)



# Plots with BayesPlot
#bayesplot::mcmc_combo(chainArray)
bayesplot::mcmc_acf(chainArray)
```



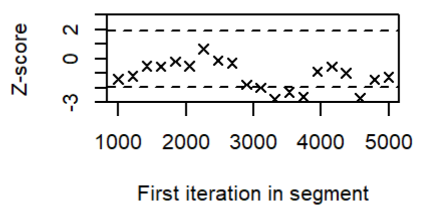
```
# Diagnostic with coda
coda.fit <- as.mcmc(modLogit.fit)

coda::geweke.plot(coda.fit)
```



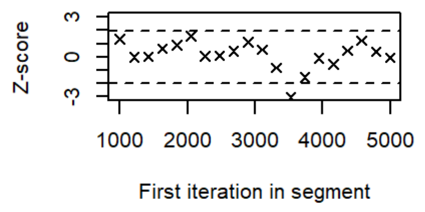
deviance (chain1)



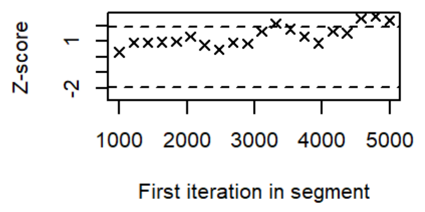
theta[1] (chain1)



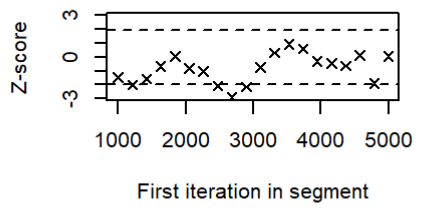
theta[2] (chain1)



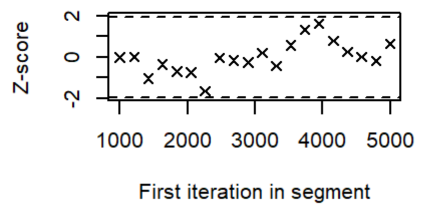
theta[3] (chain1)



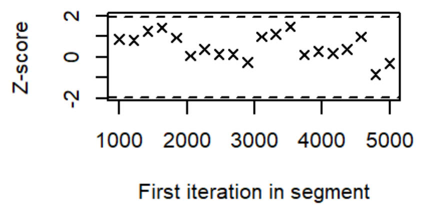
theta[4] (chain1)



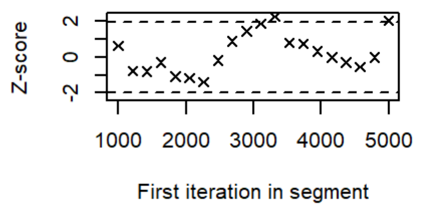
theta[5] (chain1)



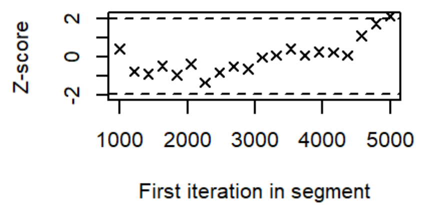
theta[6] (chain1)



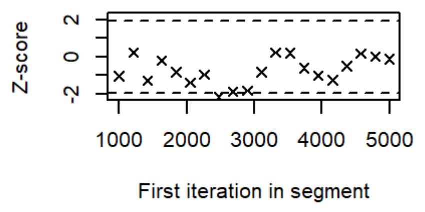
deviance (chain2)



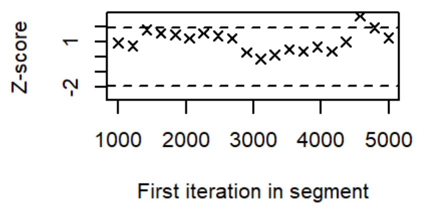
theta[1] (chain2)



theta[2] (chain2)



theta[3] (chain2)

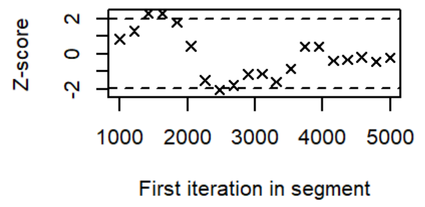


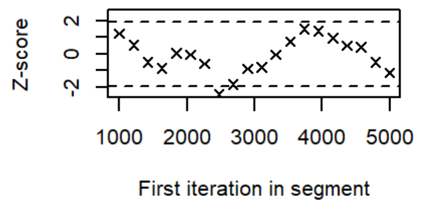
theta[4] (chain2)

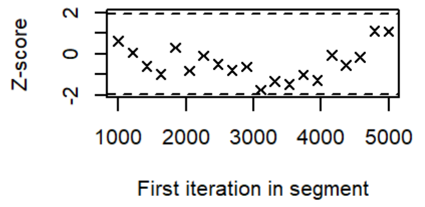
theta[5] (chain2)

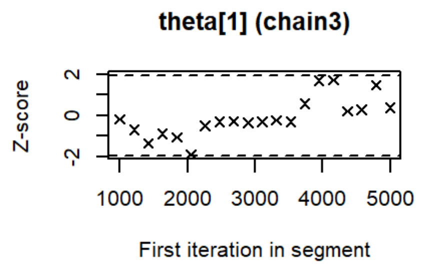
theta[6] (chain2)

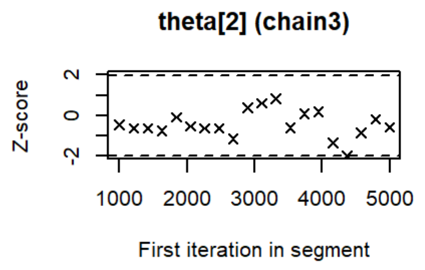
deviance (chain3)

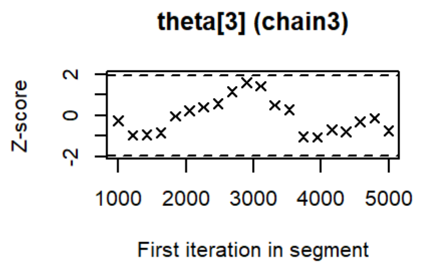

Z-score
2
0
-2
1000 2000 3000 4000 5000
First iteration in segment

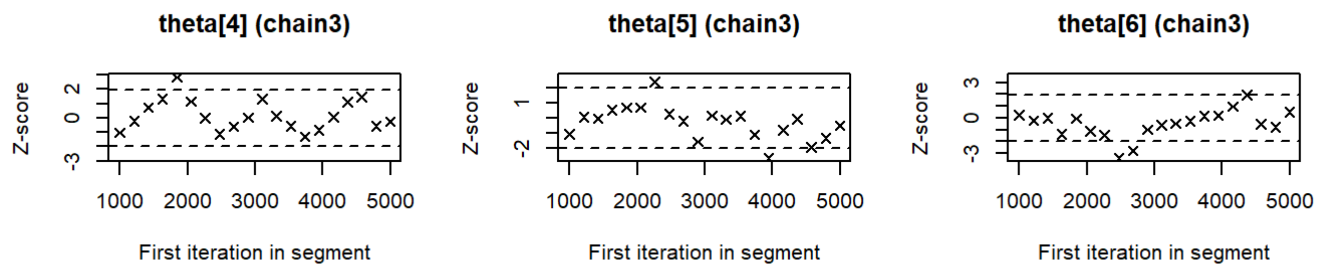

Z-score
2
0
-2
1000 2000 3000 4000 5000
First iteration in segment


Z-score
2
0
-2
1000 2000 3000 4000 5000
First iteration in segment

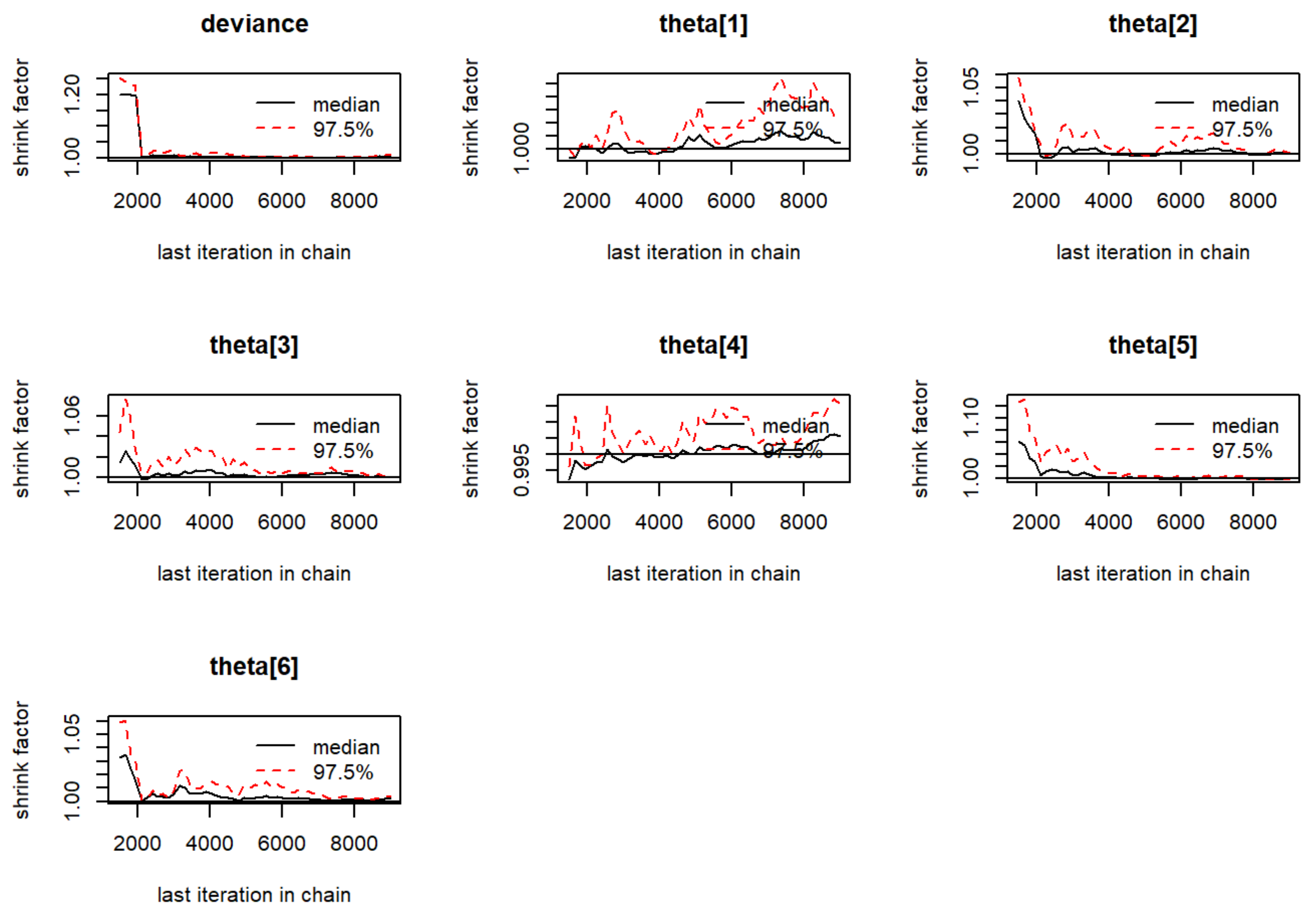

theta[1] (chain3)
Z-score
2
0
-2
1000 2000 3000 4000 5000
First iteration in segment


theta[2] (chain3)
Z-score
2
0
-2
1000 2000 3000 4000 5000
First iteration in segment


theta[3] (chain3)
Z-score
2
0
-2
1000 2000 3000 4000 5000
First iteration in segment



```
coda::gelman.plot(coda.fit)
```



```
# we can see that the chains are good,although perhaps not optimal.


# Manipulating the chain ----------------------------------------------------------------

chainMat <- modLogit.fit$BUGSoutput$sims.matrix

# Point estimates
(theta.hat.jags <- colMeans(chainMat))
```

```
##    deviance    theta[1]    theta[2]    theta[3]    theta[4]    theta[5] 
## 737.8351975  -0.8823168  -0.7979050   0.4233801   0.4912432   0.4885728 
##    theta[6] 
##   0.8516612
```

```
# Intervals
cred <- 0.95
(theta.ET.jags <- apply(chainMat, 2, quantile, prob=c((1-cred)/2, 1-(1-cred)/2)))
```

```
##       deviance   theta[1]   theta[2]  theta[3]  theta[4]  theta[5]
## 2.5%  732.9314 -1.0805478 -0.9530941 0.2887280 0.3108164 0.2866922
## 97.5% 745.7742 -0.6932779 -0.6548805 0.5609768 0.6761416 0.6931394
##        theta[6]
## 2.5%  0.6129723
## 97.5% 1.1046229
```

```
# What about the HPD?
(theta.HPD.jags <- coda::HPDinterval(as.mcmc(chainMat)))
```

```
##                lower       upper
## deviance 732.1230582 744.0519188
## theta[1]  -1.0598029  -0.6866232
## theta[2]  -0.9566374  -0.6595035
## theta[3]   0.2814150   0.5531108
## theta[4]   0.3133497   0.6769011
## theta[5]   0.2997378   0.7024840
## theta[6]   0.6243411   1.1123852
## attr(,"Probability")
## [1] 0.95
```

```
cat('parameters via MCMC with original data are', theta.hat.jags[-1], 'parameters via MLE are:',MLE$par)
```

```
## parameters via MCMC with original data are -0.8823168 -0.797905 0.4233801 0.4912432 0.4885728 0.8516612 parame
ters via MLE are: -0.8676375 -0.7814026 0.4191899 0.4817931 0.4308103 0.8127787
```

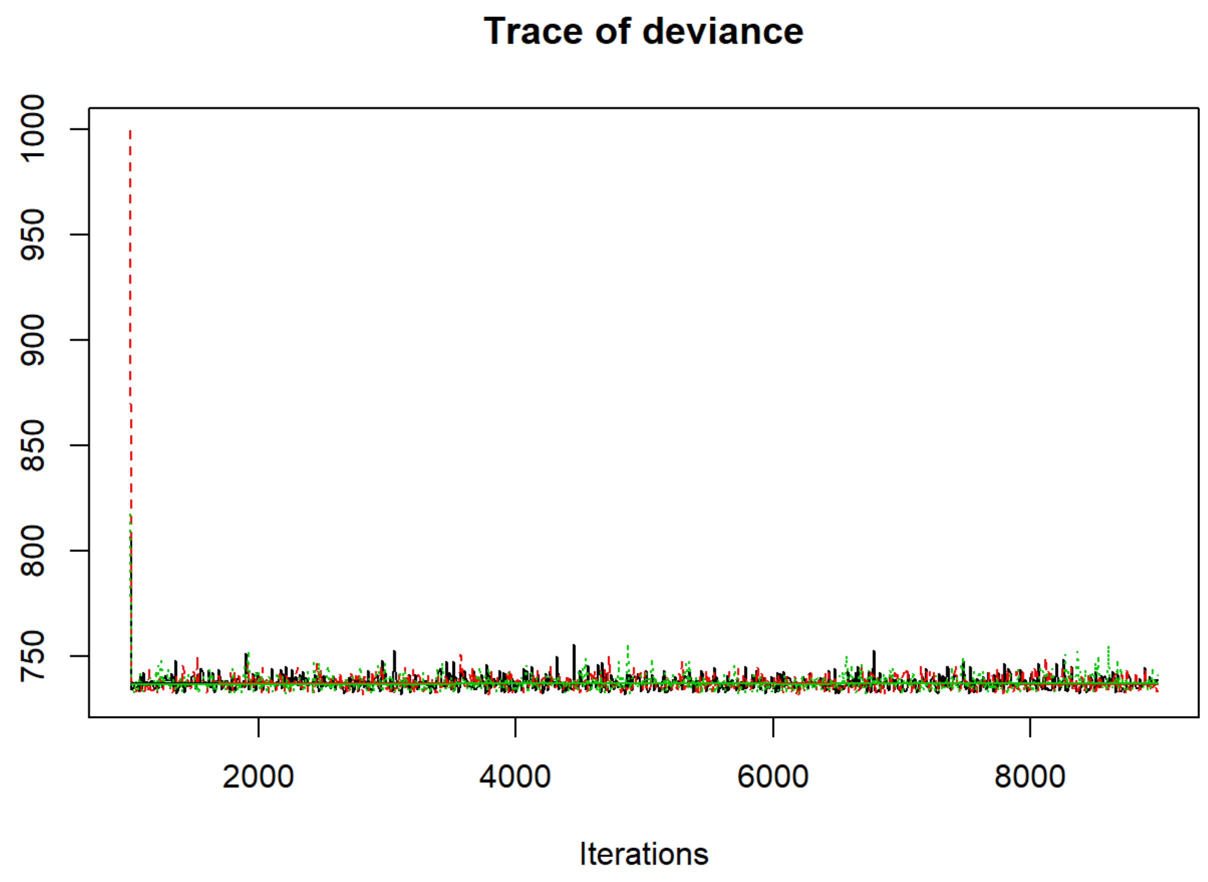
```
#Error approximation
errors=sapply(modLogit.fit$BUGSoutput$sd, function(x) x^2/(modLogit.fit$n.iter-1000))
cat('approximation errors are ',errors$theta)
```

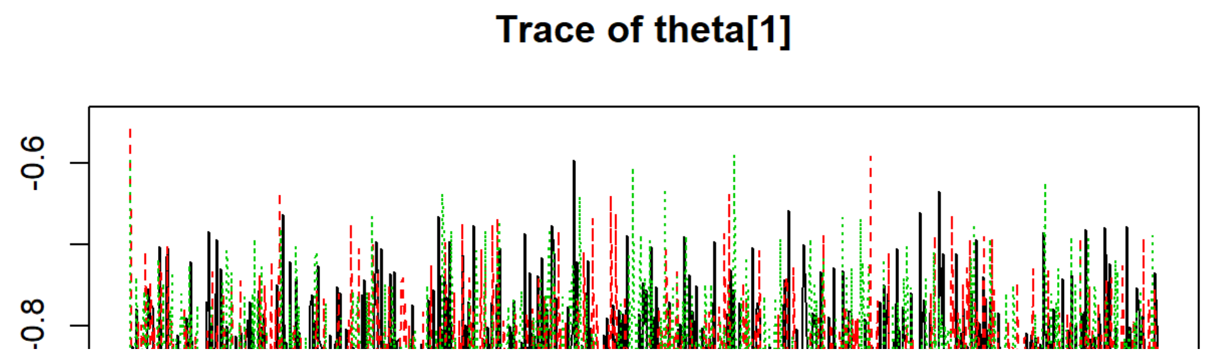
```
## approximation errors are  1.182868e-06 7.418513e-07 6.355698e-07 1.108487e-06 1.410216e-06 1.981918e-06
```

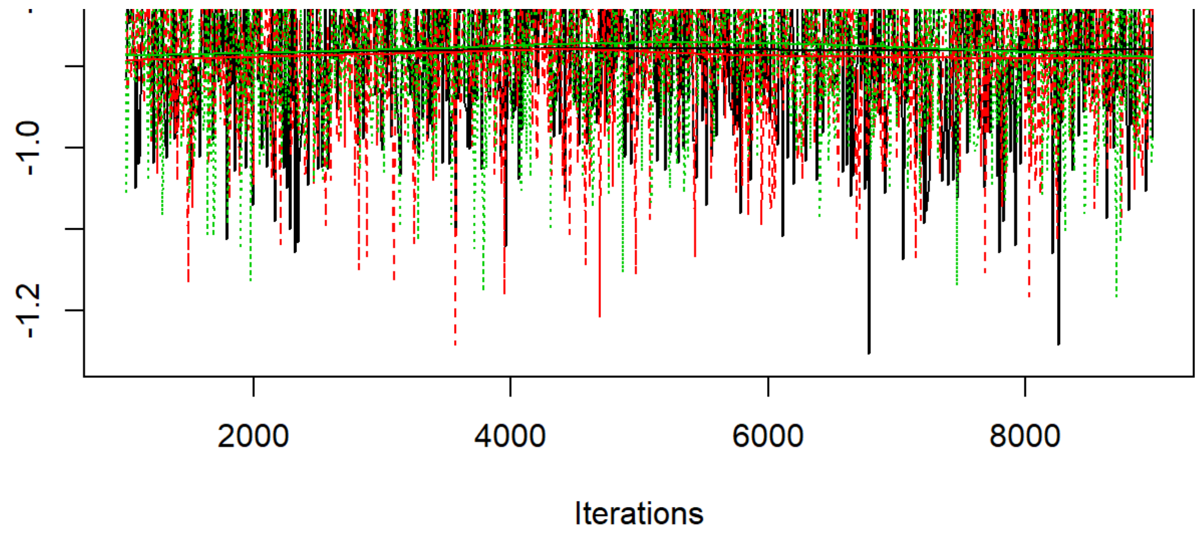
```
# Parameter uncertainty
cat('uncertainty for parameters',abs(modLogit.fit$BUGSoutput$sd$theta/theta.hat.jags[-1]))
```

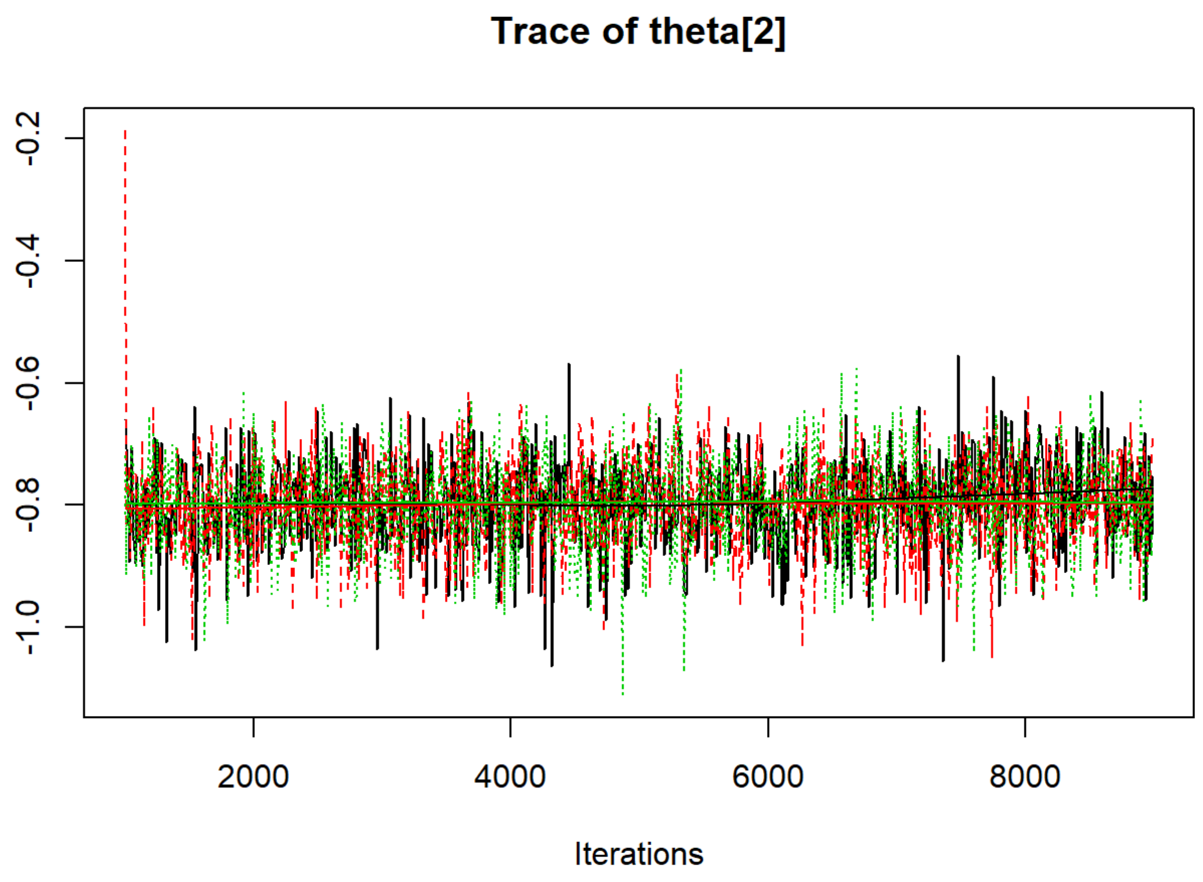
```
## uncertainty for parameters 0.1102525 0.09654999 0.168421 0.1916963 0.2173995 0.1478499
```

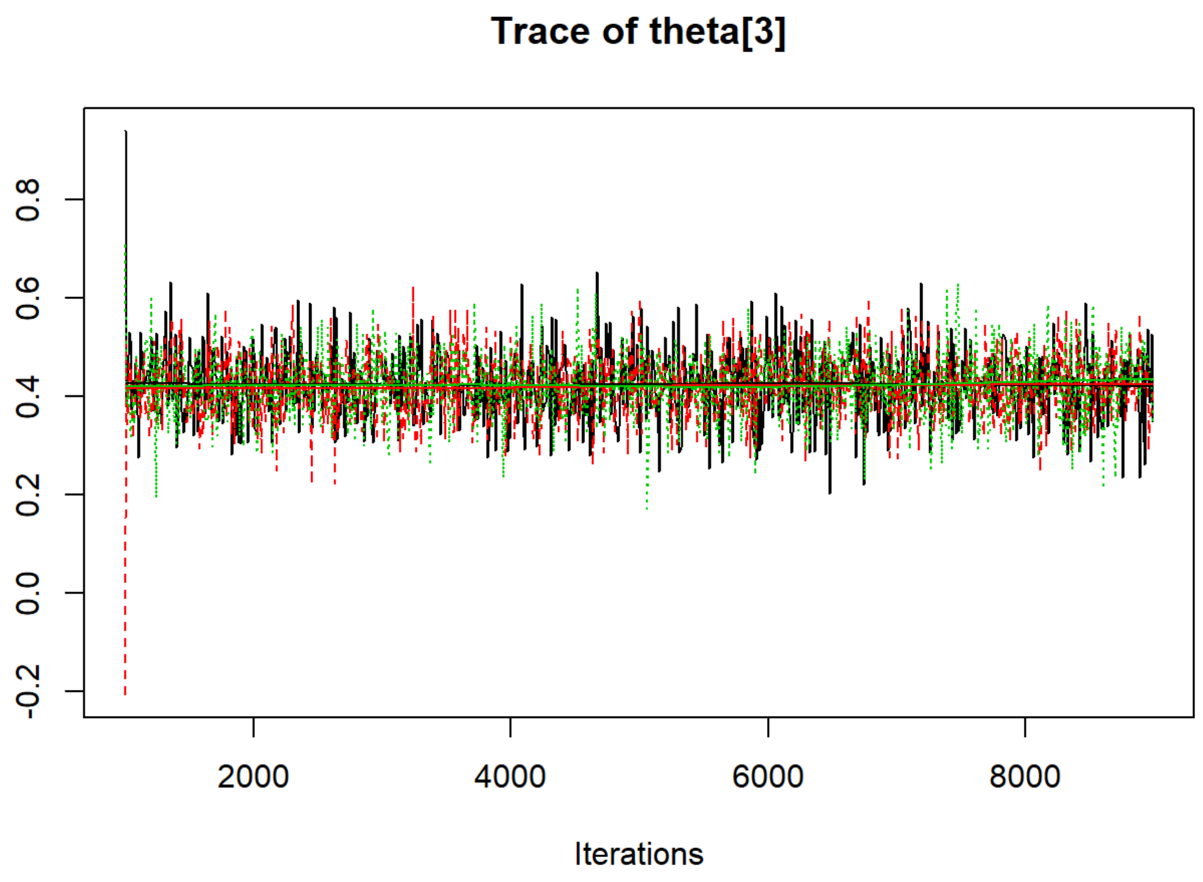
```
#Traceplot
traceplot(as.mcmc(modLogit.fit))
```

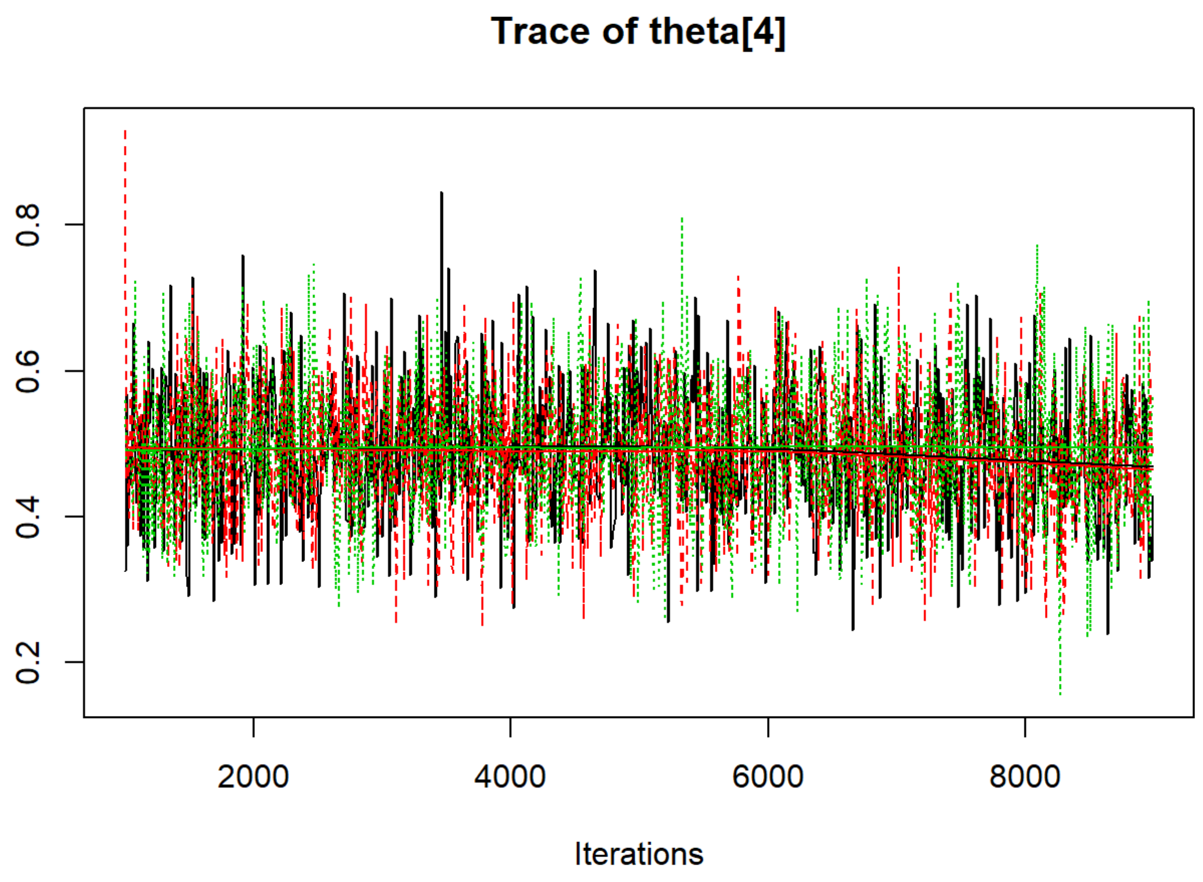

Trace of deviance
1000
950
900
850
800
750
2000
4000
6000
8000
Iterations

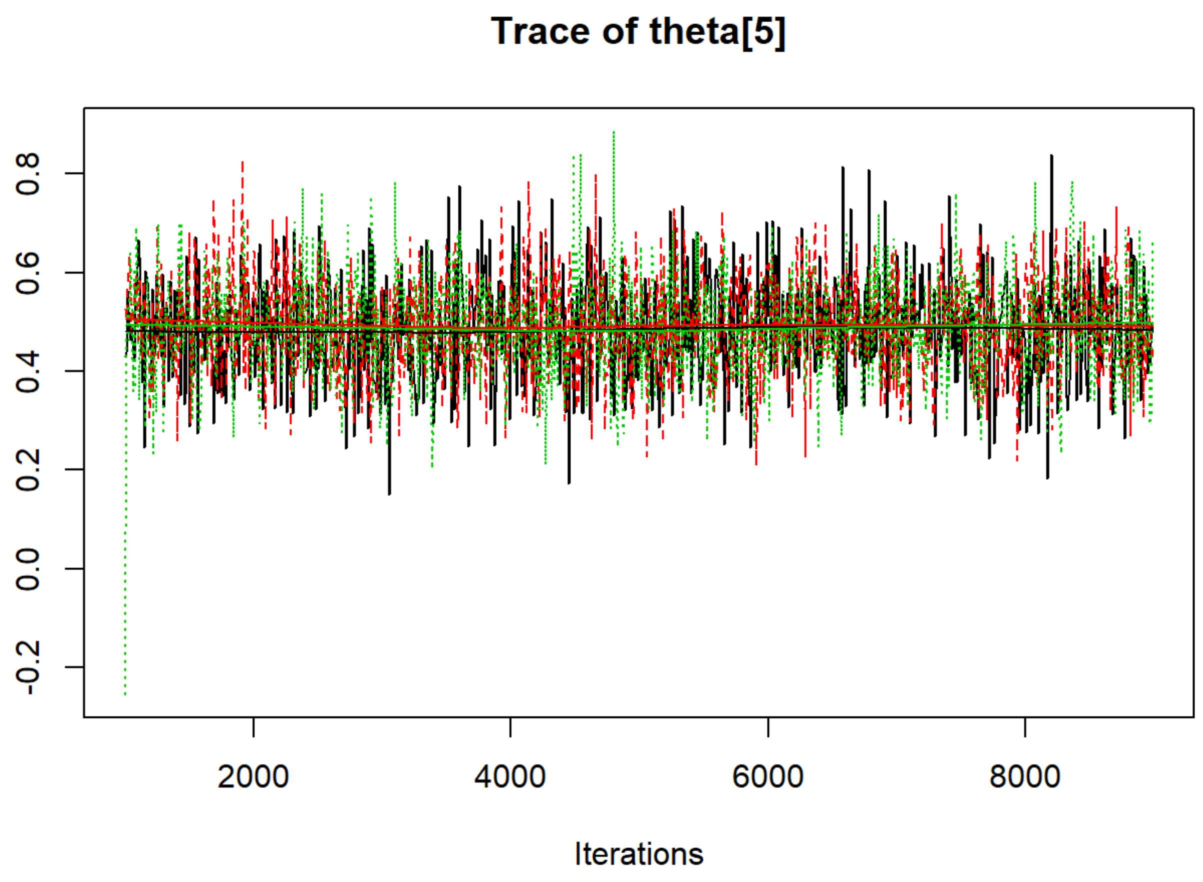

Trace of theta[1]
-0.6
-0.8

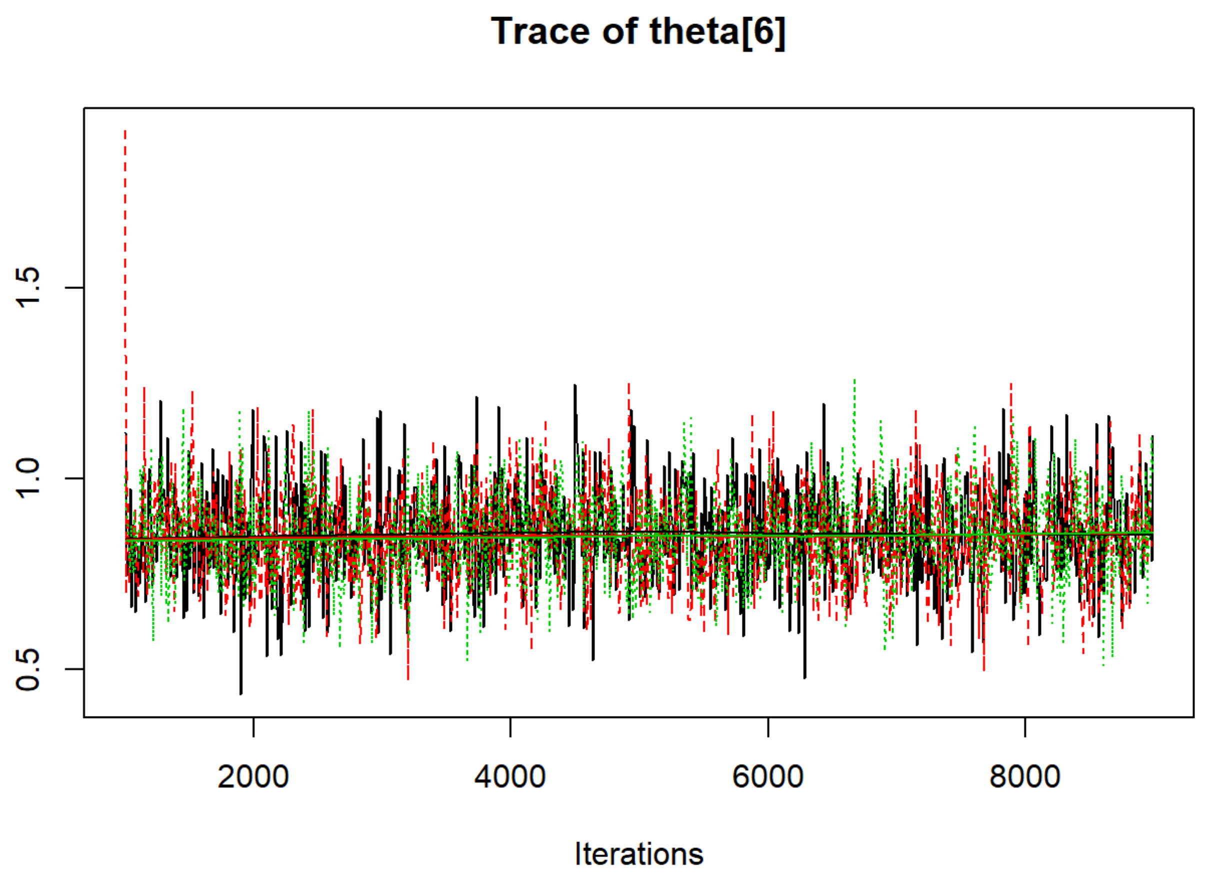

-1.0
-1.2
2000
4000
6000
8000
Iterations


Trace of theta[2]
-0.2
-0.4
-0.6
-0.8
-1.0
2000
4000
6000
8000
Iterations

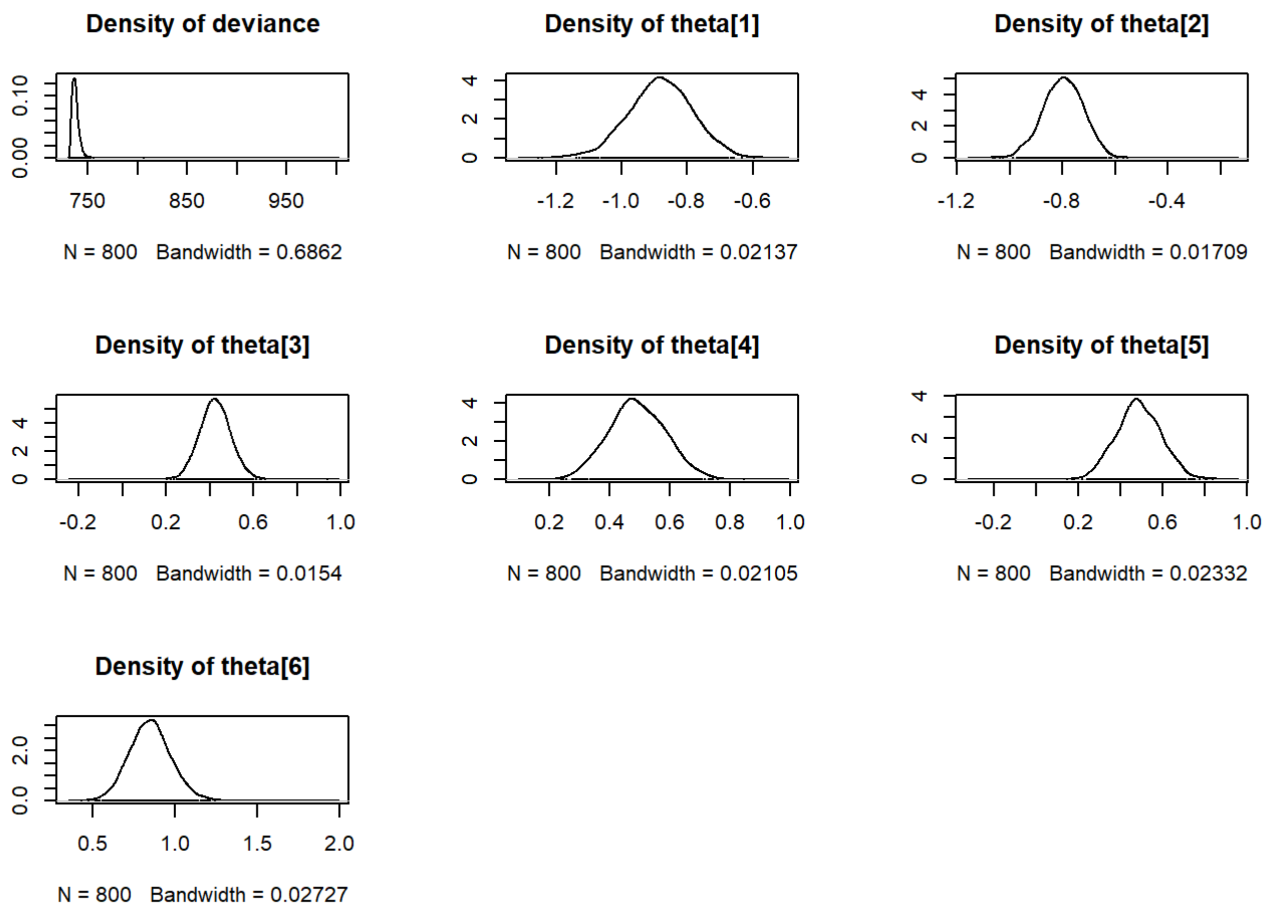

Trace of theta[3]
0.8
0.6
0.4
0.2
0.0
-0.2
2000
4000
6000
8000
Iterations


Trace of theta[4]
0.2
0.4
0.6
0.8
2000
4000
6000
8000
Iterations


Trace of theta[5]
0.8
0.6
0.4
0.2
0.0
-0.2
2000
4000
6000
8000
Iterations



```
# Density plots
plot(as.mcmc(modLogit.fit),density = T,trace = F)
```



```
#Gelmna-Rubin Test

gelman.diag(as.mcmc(modLogit.fit))
```

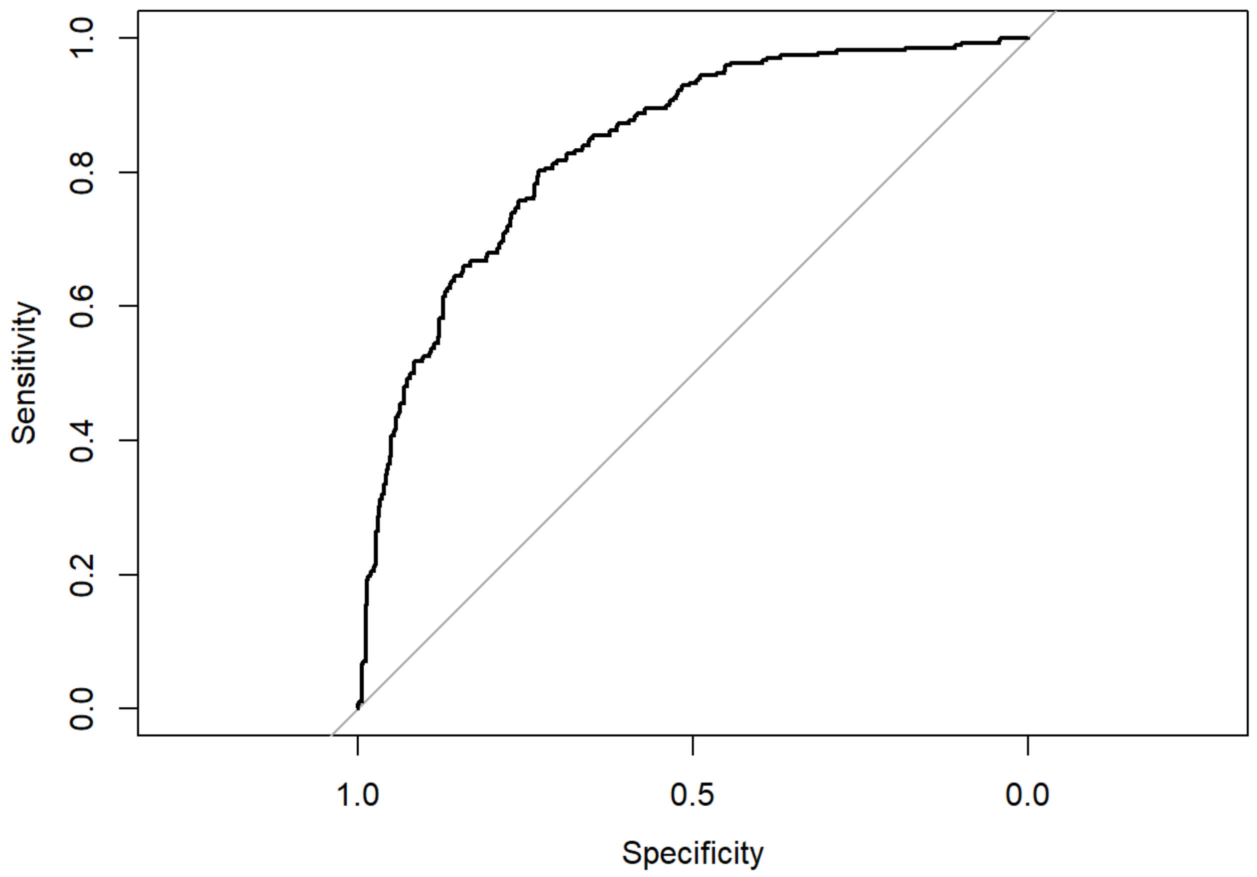
```
## Potential scale reduction factors:
##
##          Point est. Upper C.I.
## deviance      1.005      1.009
## theta[1]      1.002      1.010
## theta[2]      1.001      1.001
## theta[3]      1.000      1.001
## theta[4]      1.006      1.016
## theta[5]      0.999      0.999
## theta[6]      1.002      1.004
##
## Multivariate psrf
##
## 1.01
```

```
#gelman.plot(as.mcmc(modLogit.fit))

#Heidelberger-Welch Test
heidel.diag(as.mcmc(modLogit.fit))
```

```
## [[1]]
##
##          Stationarity start     p-value
##          test         iteration
## deviance passed       81        0.12858
## theta[1] passed        1        0.75123
## theta[2] failed       NA        0.00781
## theta[3] passed        1        0.37639
## theta[4] failed       NA        0.00413
## theta[5] passed        1        0.27680
## theta[6] passed        1        0.70495
##
##          Halfwidth Mean    Halfwidth
##          test
## deviance passed    737.727 0.21103
## theta[1] passed     -0.881 0.00662
## theta[2] <NA>           NA      NA
## theta[3] passed      0.426 0.00506
## theta[4] <NA>           NA      NA
## theta[5] passed      0.484 0.00736
## theta[6] passed      0.855 0.00806
```

```
##
## [[2]]
##
##          Stationarity start     p-value
##          test         iteration
## deviance passed        81       0.4739
## theta[1] passed         1       0.7333
## theta[2] passed         1       0.6939
## theta[3] passed        81       0.0861
## theta[4] passed       161       0.0719
## theta[5] passed         1       0.6677
## theta[6] passed         1       0.8560
##
##          Halfwidth Mean    Halfwidth
##          test
## deviance passed    737.626 0.23619
## theta[1] passed     -0.888 0.00590
## theta[2] passed     -0.799 0.00525
## theta[3] passed      0.422 0.00433
## theta[4] passed      0.484 0.00632
## theta[5] passed      0.493 0.00720
## theta[6] passed      0.851 0.00907
##
## [[3]]
##
##          Stationarity start     p-value
##          test         iteration
## deviance passed       1         0.494
## theta[1] passed       1         0.290
## theta[2] passed       1         0.759
## theta[3] passed       1         0.703
## theta[4] passed       1         0.978
## theta[5] passed       1         0.465
## theta[6] passed       1         0.409
##
##          Halfwidth Mean    Halfwidth
##          test
## deviance passed    737.799 0.31777
## theta[1] passed     -0.878 0.00679
## theta[2] passed     -0.797 0.00513
## theta[3] passed      0.424 0.00484
## theta[4] passed      0.496 0.00637
## theta[5] passed      0.488 0.00751
## theta[6] passed      0.849 0.00873
```

```
#ROC curve of point estimates
fit_values_MC=theta.hat.jags[-1][1]+theta.hat.jags[-1][2]*X[,1]+theta.hat.jags[-1][3]*X[,2]+theta.hat.jags[-1][4]
*X[,3]+theta.hat.jags[-1][5]*X[,4]+theta.hat.jags[-1][6]*X[,5]
roc(Y,fit_values_MC,plot=T)
```



```
##
## Call:
## roc.default(response = Y, predictor = fit_values_MC, plot = T)
##
## Data: fit_values_MC in 500 controls (Y 0) < 268 cases (Y 1).
## Area under the curve: 0.8363
```

The code above creates and simulates the Markov Chains and outputs their diagnostics,both visually and numerically. What concerns us first is to see whether the chains converge to a stationary distribution.Based on the R-hat values of the parameters,but also the traceplots and the autocorrelation plots, we can make this assumption.In order to check the convergence more formaly, we performed 2 type of tests, Gelman-Rubin and Heidelberger-Welch.

Gelman-Rubin Test

Synoptically,the Gelman-Rubin test checks convergence of MCMC output in which m>1 parallel chains are run with starting values that are overdispersed relative to the posterior distribution. Convergence is diagnosed when the chains have `forgotten' their initial values, and the output from all chains is indistinguishable. It is based a comparison of within-chain and between-chain variances, and is similar to a classical analysis of variance.The convergence diagnostic itself is a value for each parameter,with values substansially above 1 indicating lack of convergence.

Heidelberger-Welch Test

The Heidelberger-Welch test consists of 2 parts.The idea is to perfor multiple hypothesis in the chains to check if they have converged to the stationary distribution.It has the capability to discard a certain percentage of the chain and then re-perform the hypothesis test in the rest of tha chain.As far as the second part concerned,the ratio of half of the length of the credible interval of each chain,over its mean, is compared with a threshold value,suggested by the authors,and if it is higher than the threshold,then the chain must be updated for more iterations.

Sometimes the visual diagnostics from the coda package are not so trivial to claim or not convergence.Thus,we will be using the tests mentioned above to make sure of the convergence,and hopefully confirm the visual intuition. Our parameters pass these tests, so it confirms the visual results.Furthermore,we can observe the posterior densities of our parameters, plus their point estimates and credible intervals.We can observe that both point estimates and the intervals are similar to the ones obtained via the Frequentist approach.Lastly,we can see the ROC curve of the point estimates.The ROC curve shows the sensitivity and specificity for the binary results of our model. We can pick any point on the curve,depending on our preference and our task,by finding the adequate threshold, that will map the output of our model to 0 and 1.(0 if the probability is less than the threshold and 1 if it is larger than the threshold).

Simulate Data from Model hypothesis

Next, we will simulate the data,according to our model. More specifically, we will assume the variables follow a normal distribution,with mean=0 and sd=8. As our variables are the Principal Components,whose histograms can be found above, we will try a rather big value as standard deviation,so it includes outliers,which were not present in the original dataset.In this way, we will check the model's ability to recover its true parameters obtained with the real data.If it does, it means that both our model and our estimation technique are correct.

```
# Simulate data --------------------------------------------------------------------

# Pick sample size
N <- 1000

X1 <- rnorm(N,0,8)
```

```r
X2 <- rnorm(N,0,8)
X3 <- rnorm(N,0,8)
X4 <- rnorm(N,0,8)
X5 <- rnorm(N,0,8)
# Since our variables are the scaled PCA components, the standard deviation of 8 is quite big and could include o
utliers.


# Pick fixed values for the parameters of the model
theta=c(-0.8823168, -0.797905, 0.4233801, 0.4912432, 0.4885728, 0.8516612)

# Simulate response according to the model
linpred <- theta[1] + theta[2]*X1 +theta[3]*X2+theta[4]*X3+theta[5]*X4+theta[6]*X5
pis <- exp(linpred)/(1+exp(linpred))

y=replicate(N,NA)
for(i in 1:N){
  y[i]=rbern(1,pis[i])
}
#y<-rbern(N,pis)

dat <- data.frame(X1=X1, X2=X2,X3=X3,X4=X4,X5=X5,y=y)




# MCMC inference ----------------------------------------------------------


modelLogitSim <- function() {
  # Likelihood
  for(i in 1:n)
  {
    y[i] ~ dbern(p[i])

    logit(p[i]) <- theta[1] + theta[2]*X1[i] +theta[3]*X2[i]+theta[4]*X3[i]+theta[5]*X4[i]+theta[6]*X5[i]
  }
  # Priors
  theta[1] ~ dnorm(0, 1.0E-6)
  for (i in 2:n_param)
  {
    theta[i] ~ dnorm(0, 1.0E-6)

  }
}

  # Preparing data for JAGS -------------------------------------------------
X1=dat$X1
X2=dat$X2
X3=dat$X3
X4=dat$X4
X5=dat$X5
y=dat$y
n=nrow(dat)
n_param=ncol(dat)

dat.jags <- list("X1", "X2", "X3", "X4", "X5", "y", "n","n_param")


# Parameters --------------------------------------------------------------

modLogitSim.params="theta"

# Starting values
modLogitSim.inits <- function(){
  list("theta" = c(rnorm(n_param, 0, 0.1))
  )
}



set.seed(123)
modLogitSim.fit <- jags(data = dat.jags,                                    # DATA
                      model.file = modelLogitSim, inits = modLogitSim.inits,          # MODEL
                      parameters.to.save = modLogitSim.params,
                      n.chains = 3, n.iter = 9000, n.burnin = 1000, n.thin=10)      # MCMC
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1000
##    Unobserved stochastic nodes: 6
##    Total graph size: 13010
##
## Initializing model
```

```
modLogitSim.fit
```

```
## Inference for Bugs model at "C:/Users/teoso/AppData/Local/Temp/RtmpCKRd8h/model31c8c3aad4.txt", fit using jag
s,
##  3 chains, each with 9000 iterations (first 1000 discarded), n.thin = 10
##  n.sims = 2400 iterations saved
##          mu.vect sd.vect    2.5%     25%     50%     75%   97.5%  Rhat
## theta[1]  -1.150   0.232  -1.594  -1.302  -1.145  -0.999  -0.713 1.024
## theta[2]  -0.876   0.102  -1.069  -0.932  -0.877  -0.820  -0.688 1.073
## theta[3]   0.473   0.057   0.368   0.441   0.472   0.507   0.585 1.053
## theta[4]   0.542   0.064   0.424   0.505   0.541   0.578   0.668 1.039
## theta[5]   0.543   0.066   0.423   0.507   0.542   0.582   0.670 1.058
## theta[6]   0.975   0.111   0.765   0.915   0.975   1.039   1.182 1.046
## deviance 201.100  14.441 195.391 197.640 199.650 202.311 210.865 1.004
##          n.eff
## theta[1]    94
## theta[2]    38
## theta[3]    70
## theta[4]    61
## theta[5]    79
## theta[6]    56
## deviance  1400
```

```
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 104.3 and DIC = 305.4
## DIC is an estimate of expected predictive error (lower deviance is better).
```

```
# Results and diagnostics ------------------------------------------------

modLogitSim.fit
```

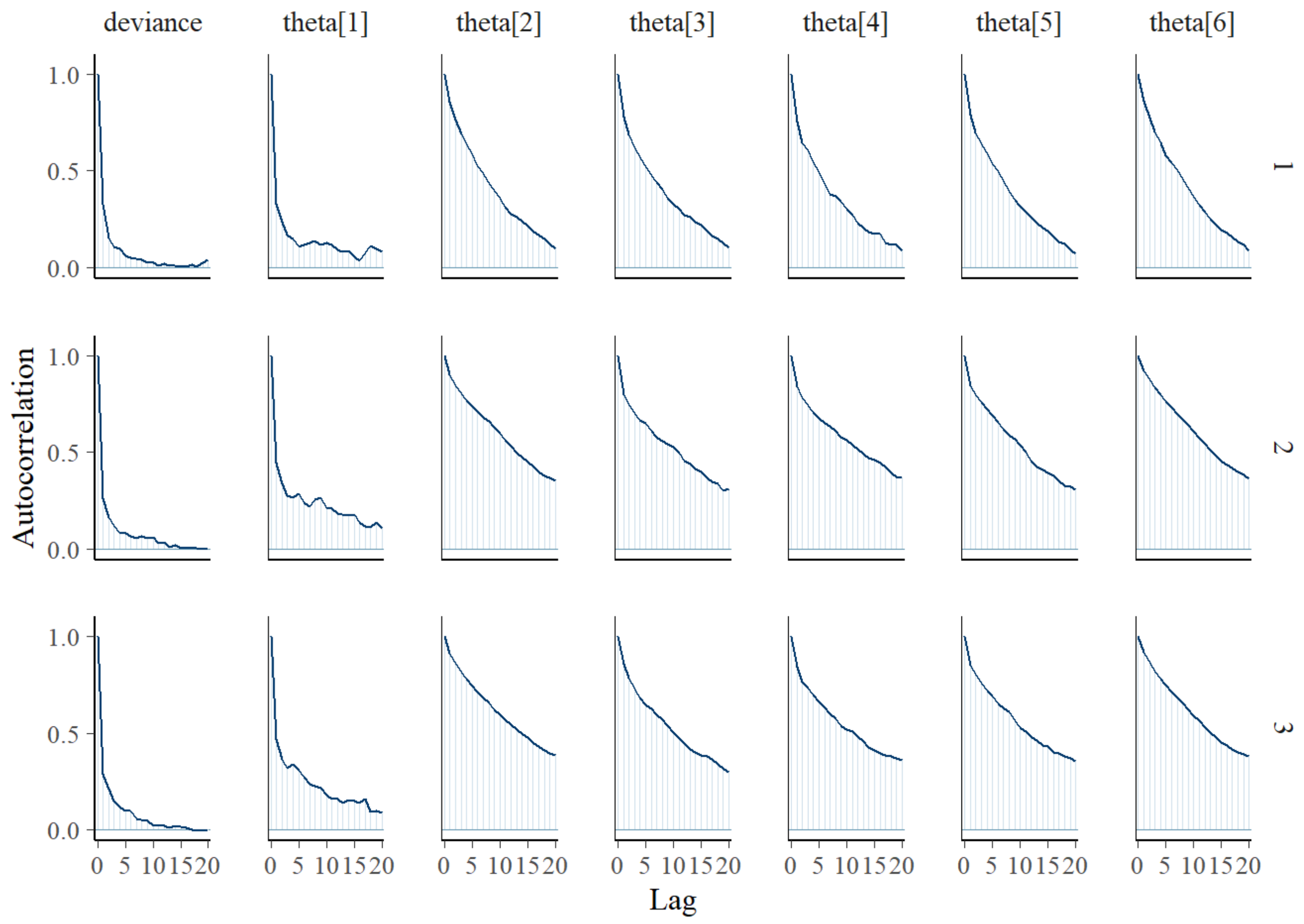
```
## Inference for Bugs model at "C:/Users/teoso/AppData/Local/Temp/RtmpCKRd8h/model31c8c3aad4.txt", fit using jag
s,
##  3 chains, each with 9000 iterations (first 1000 discarded), n.thin = 10
##  n.sims = 2400 iterations saved
##          mu.vect sd.vect    2.5%     25%     50%     75%   97.5%  Rhat
## theta[1]  -1.150   0.232  -1.594  -1.302  -1.145  -0.999  -0.713 1.024
## theta[2]  -0.876   0.102  -1.069  -0.932  -0.877  -0.820  -0.688 1.073
## theta[3]   0.473   0.057   0.368   0.441   0.472   0.507   0.585 1.053
## theta[4]   0.542   0.064   0.424   0.505   0.541   0.578   0.668 1.039
## theta[5]   0.543   0.066   0.423   0.507   0.542   0.582   0.670 1.058
## theta[6]   0.975   0.111   0.765   0.915   0.975   1.039   1.182 1.046
## deviance 201.100  14.441 195.391 197.640 199.650 202.311 210.865 1.004
##          n.eff
## theta[1]    94
## theta[2]    38
## theta[3]    70
## theta[4]    61
## theta[5]    79
## theta[6]    56
## deviance  1400
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
```

```
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 104.3 and DIC = 305.4
## DIC is an estimate of expected predictive error (lower deviance is better).
```

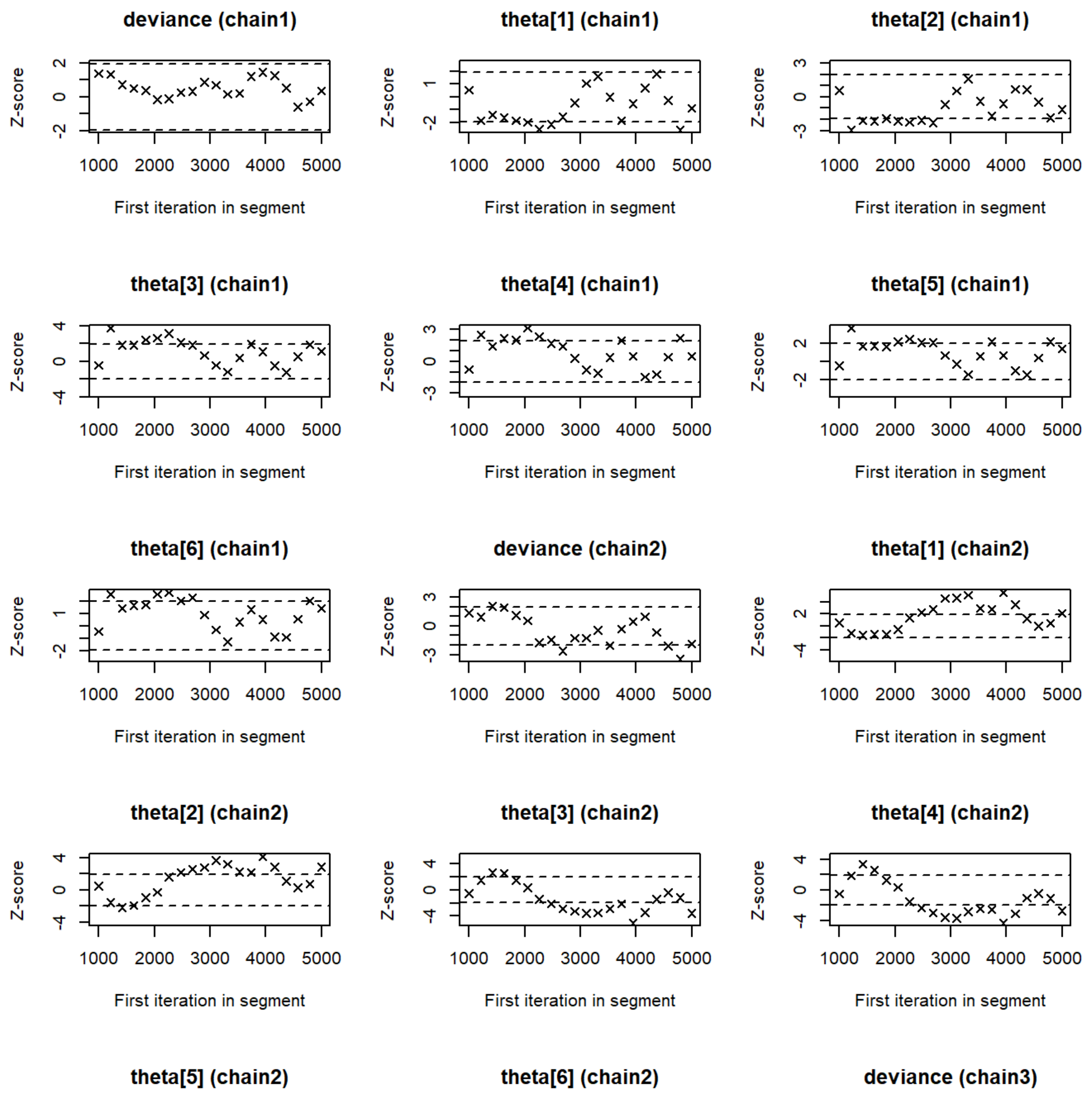
```
modLogitSim.fit$BUGSoutput$summary
```

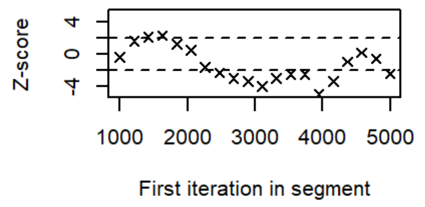
```
##                 mean          sd        2.5%         25%         50%
## deviance 201.1003847 14.44134355 195.3914332 197.6396738 199.6500896
## theta[1]  -1.1496401  0.23163931  -1.5935601  -1.3020872  -1.1445968
## theta[2]  -0.8756016  0.10200249  -1.0692056  -0.9315593  -0.8771713
## theta[3]   0.4728486  0.05748124   0.3678726   0.4410010   0.4722571
## theta[4]   0.5415236  0.06384533   0.4236842   0.5046141   0.5408145
## theta[5]   0.5430734  0.06576419   0.4225490   0.5066127   0.5416934
## theta[6]   0.9753095  0.11088091   0.7646397   0.9146860   0.9748007
##                  75%       97.5%     Rhat n.eff
## deviance 202.3105648 210.8650972 1.003767  1400
## theta[1]  -0.9990995  -0.7130076 1.024359    94
## theta[2]  -0.8200541  -0.6877775 1.073460    38
## theta[3]   0.5070115   0.5853288 1.052747    70
## theta[4]   0.5783900   0.6677612 1.038721    61
## theta[5]   0.5815924   0.6697842 1.057888    79
## theta[6]   1.0392791   1.1823354 1.046284    56
```

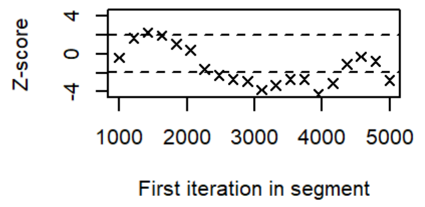
```
chainArray <- modLogitSim.fit$BUGSoutput$sims.array

# Plots with BayesPlot
#bayesplot::mcmc_combo(chainArray)
bayesplot::mcmc_acf(chainArray)
```

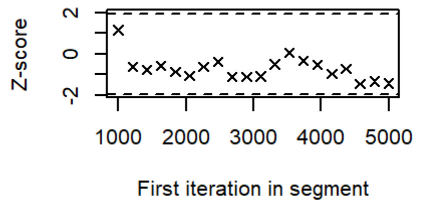


```r
# Diagnostic with coda
coda.fit <- as.mcmc(modLogitSim.fit)

coda::geweke.plot(coda.fit)
```

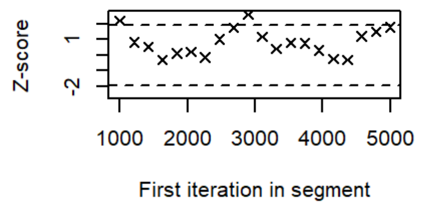

deviance (chain1)
theta[1] (chain1)
theta[2] (chain1)
theta[3] (chain1)
theta[4] (chain1)
theta[5] (chain1)
theta[6] (chain1)
deviance (chain2)
theta[1] (chain2)
theta[2] (chain2)
theta[3] (chain2)
theta[4] (chain2)
theta[5] (chain2)
theta[6] (chain2)
deviance (chain3)
Z-score
First iteration in segment
1000
2000
3000
4000
5000



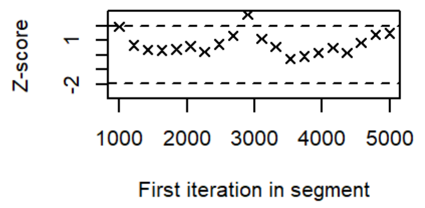




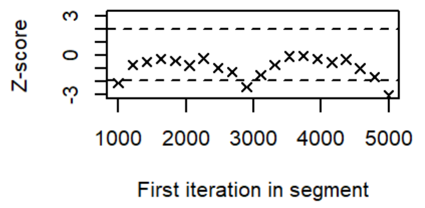
theta[1] (chain3)

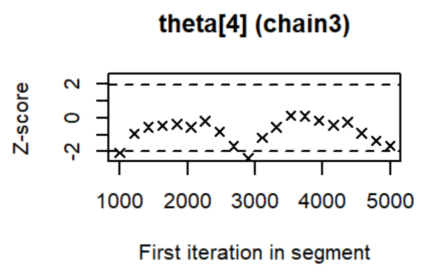


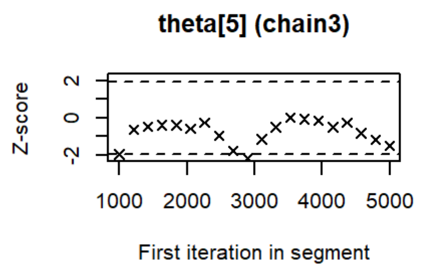
theta[2] (chain3)

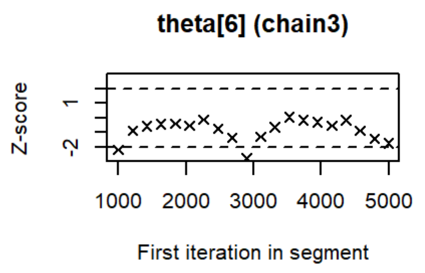


theta[3] (chain3)

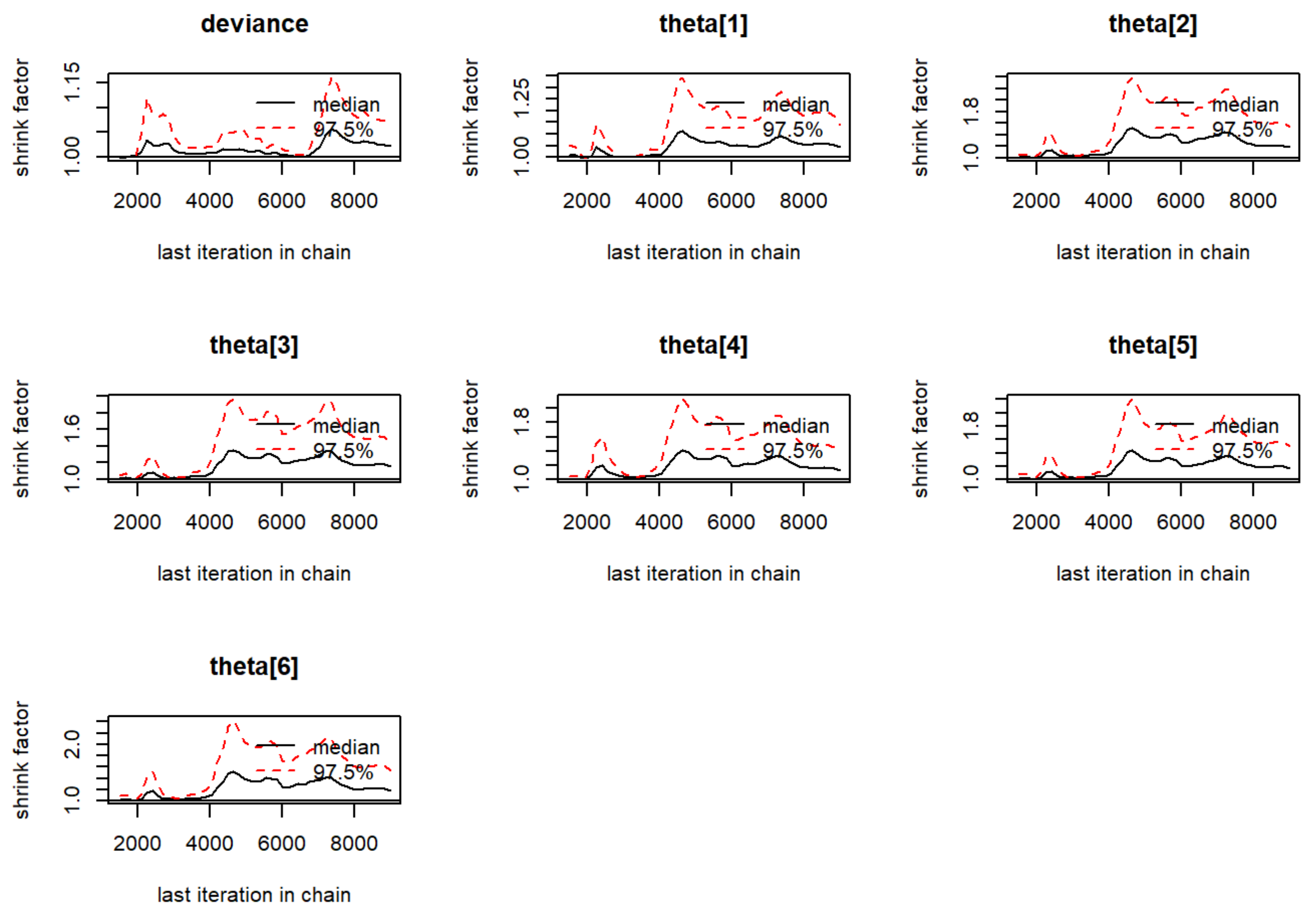








```
coda::gelman.plot(coda.fit)
```



```
# Manipulating the chain ------------------------------------------------------------

chainMat <- modLogitSim.fit$BUGSoutput$sims.matrix


# Point estimates
(theta.hat_sim.jags <- colMeans(chainMat))
```

```
##    deviance    theta[1]    theta[2]    theta[3]    theta[4]    theta[5]
## 201.1003847  -1.1496401  -0.8756016   0.4728486   0.5415236   0.5430734
##    theta[6]
##   0.9753095
```

```
# Intervals
cred <- 0.95
(theta.ET_sim.jags <- apply(chainMat, 2, quantile, prob=c((1-cred)/2, 1-(1-cred)/2)))
```

```
##       deviance   theta[1]   theta[2]  theta[3]  theta[4]  theta[5]
## 2.5%  195.3914 -1.5935601 -1.0692056 0.3678726 0.4236843 0.4225490
## 97.5% 210.8651 -0.7130076 -0.6877775 0.5853288 0.6677612 0.6697842
##        theta[6]
## 2.5%  0.7646397
## 97.5% 1.1823354
```

```
# What about the HPD?
(theta.HPD_sim.jags <- coda::HPDinterval(as.mcmc(chainMat)))
```

```
##                lower       upper
## deviance 194.7643835 207.8914072
## theta[1]  -1.5837842  -0.7049297
## theta[2]  -1.0646453  -0.6855918
## theta[3]   0.3634853   0.5799358
## theta[4]   0.4361637   0.6786094
## theta[5]   0.4282567   0.6718820
## theta[6]   0.7724413   1.1891943
## attr(,"Probability")
## [1] 0.95
```

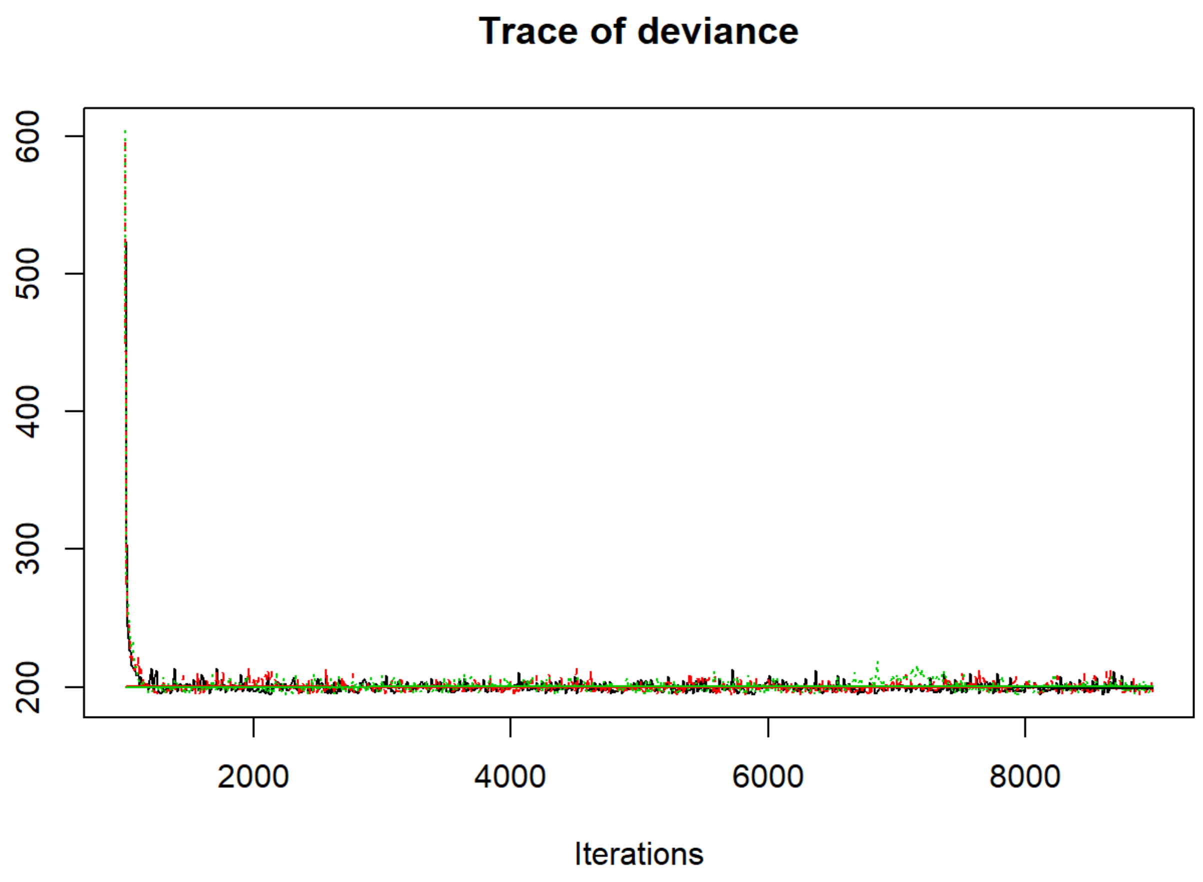
```
#Error approximation
errors=sapply(modLogitSim.fit$BUGSoutput$sd, function(x) x^2/(modLogitSim.fit$n.iter-1000))
cat('approximation errors are ',errors$theta)
```

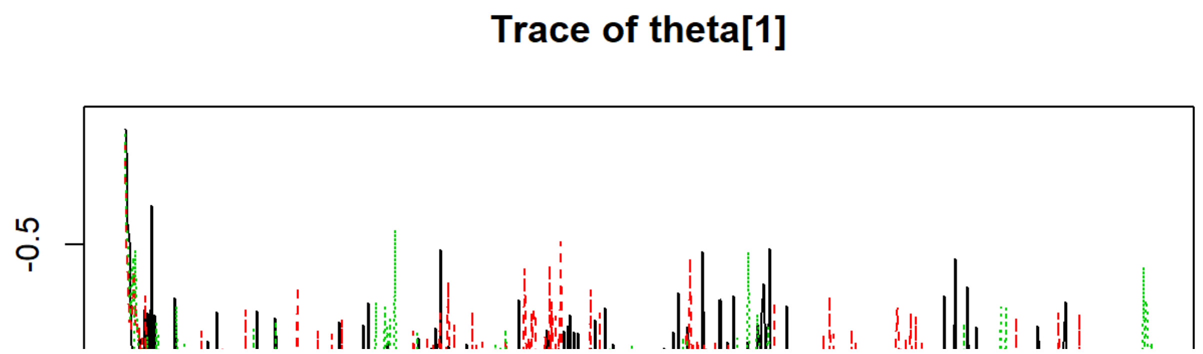
```
## approximation errors are  6.707096e-06 1.300564e-06 4.130117e-07 5.095282e-07 5.40616e-07 1.536822e-06
```

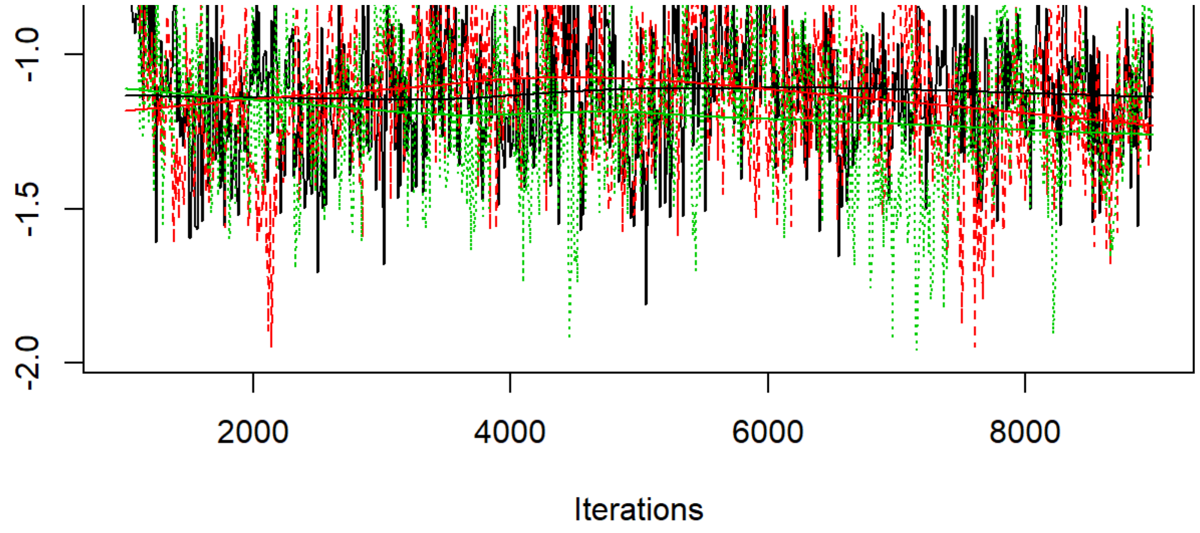
```
# Parameter uncertainty
cat('uncertainty for parameters',abs(modLogitSim.fit$BUGSoutput$sd$theta/theta.hat_sim.jags[-1]))
```

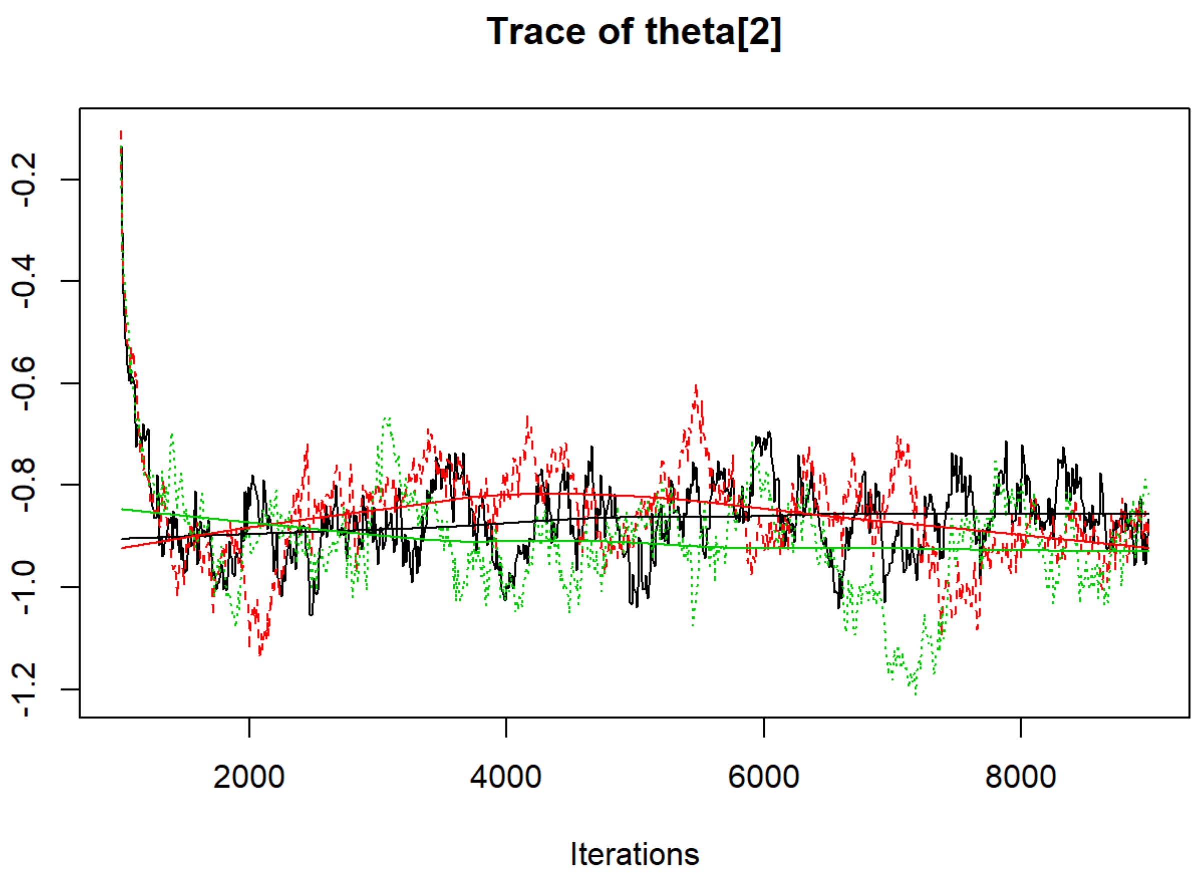
```
## uncertainty for parameters 0.2014885 0.1164942 0.1215637 0.1178994 0.1210963 0.1136879
```

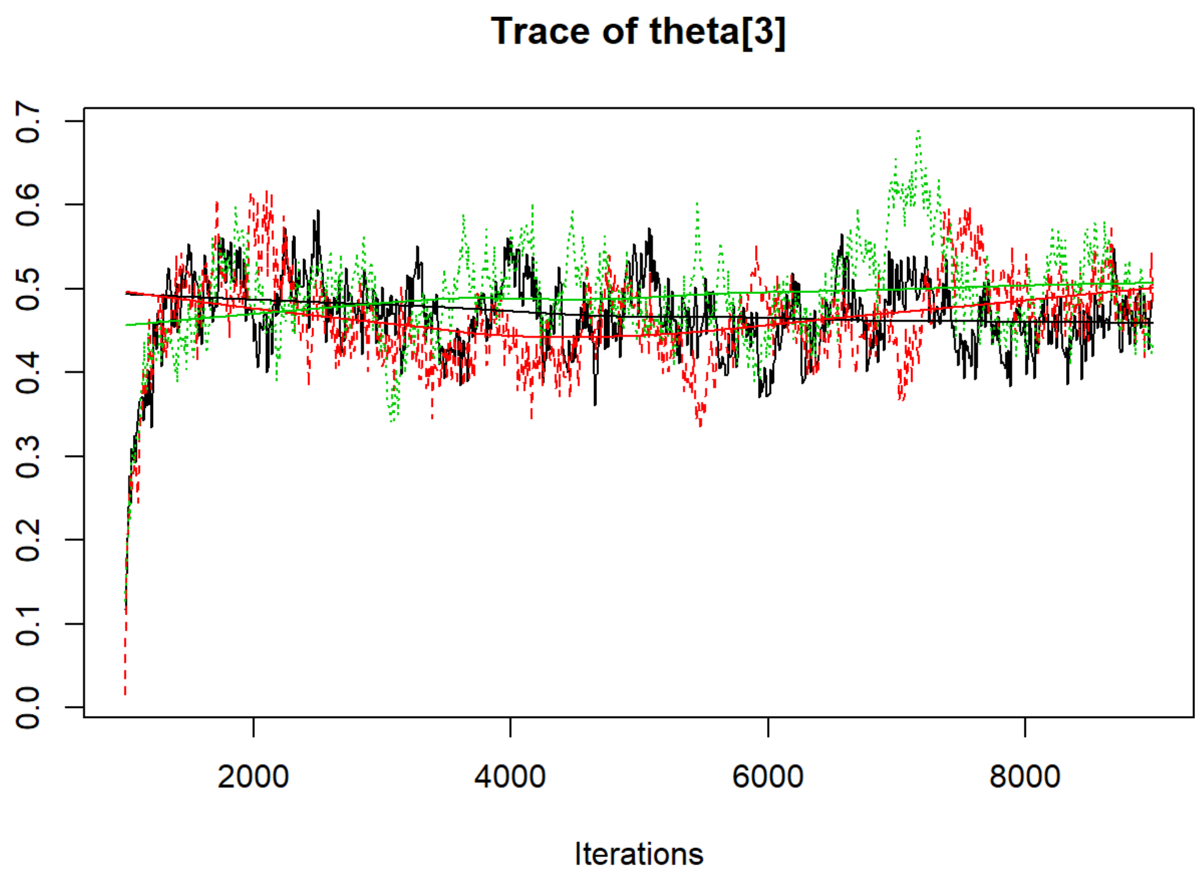
```
#Traceplot
traceplot(as.mcmc(modLogitSim.fit))
```

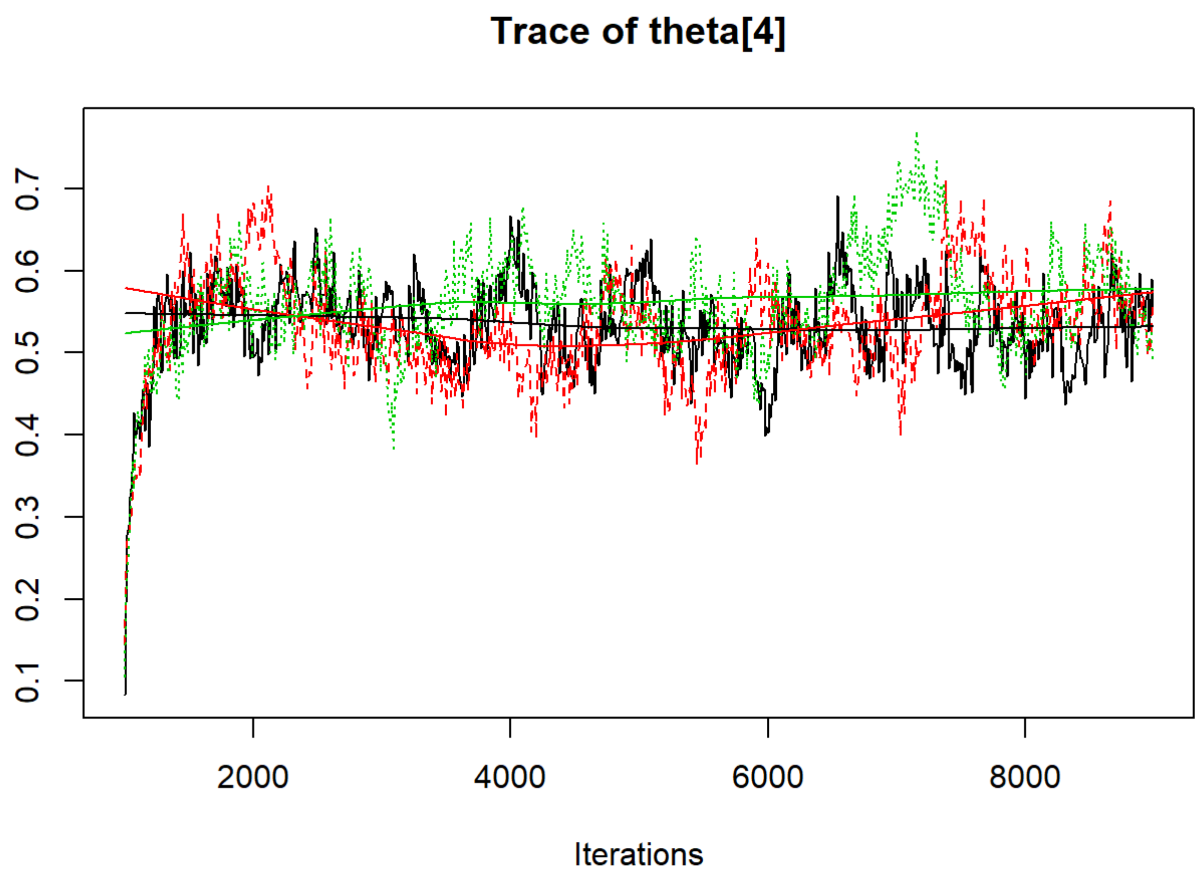

Trace of deviance
600
500
400
300
200
2000
4000
6000
8000
Iterations

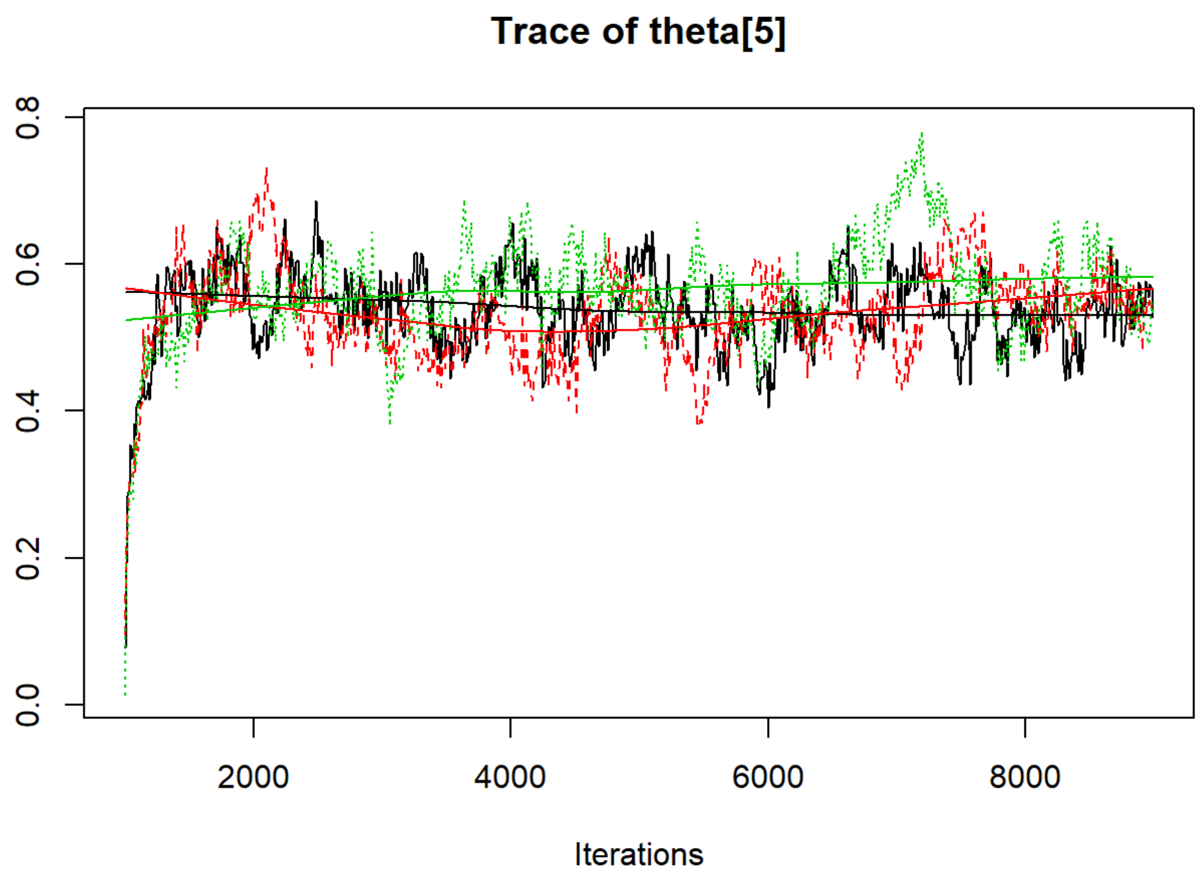

Trace of theta[1]
-0.5

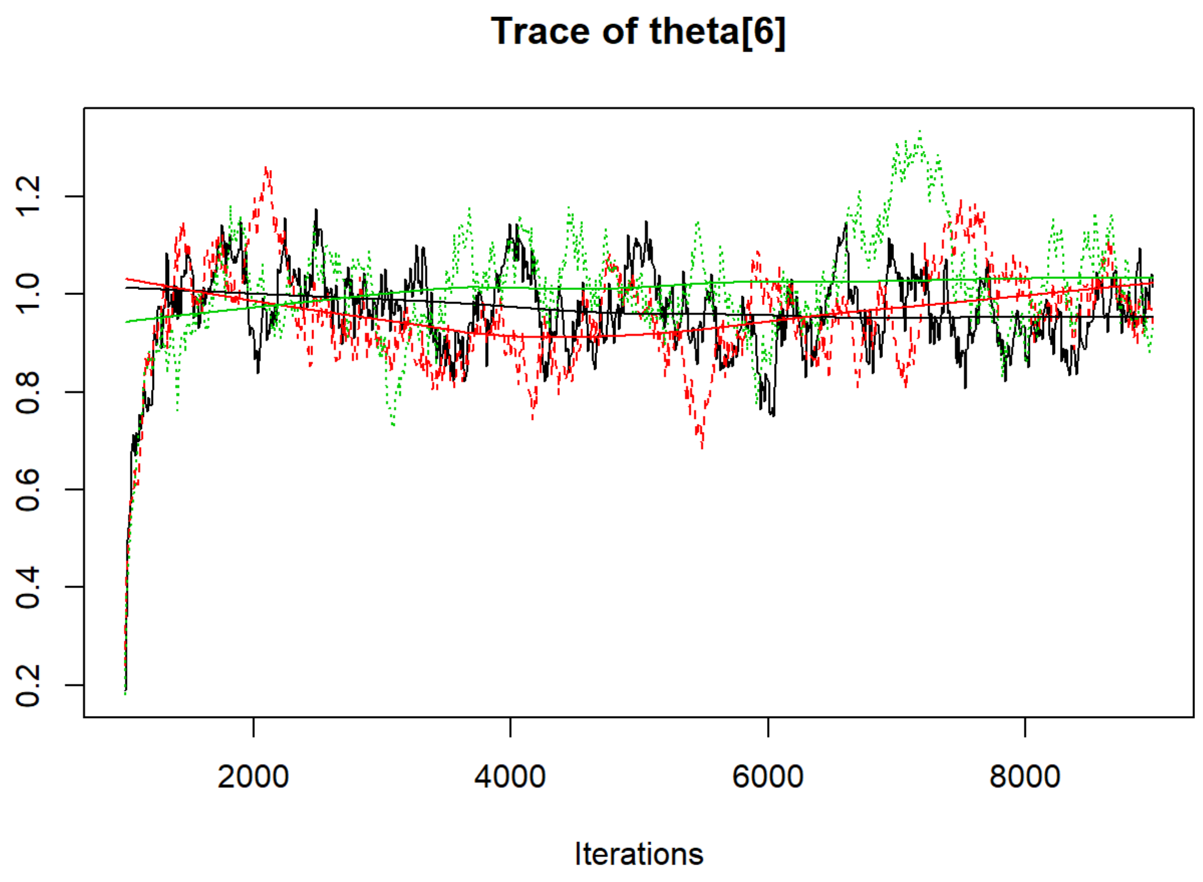

-1.0
-1.5
-2.0
2000
4000
6000
8000
Iterations


Trace of theta[2]
-0.2
-0.4
-0.6
-0.8
-1.0
-1.2
2000
4000
6000
8000
Iterations

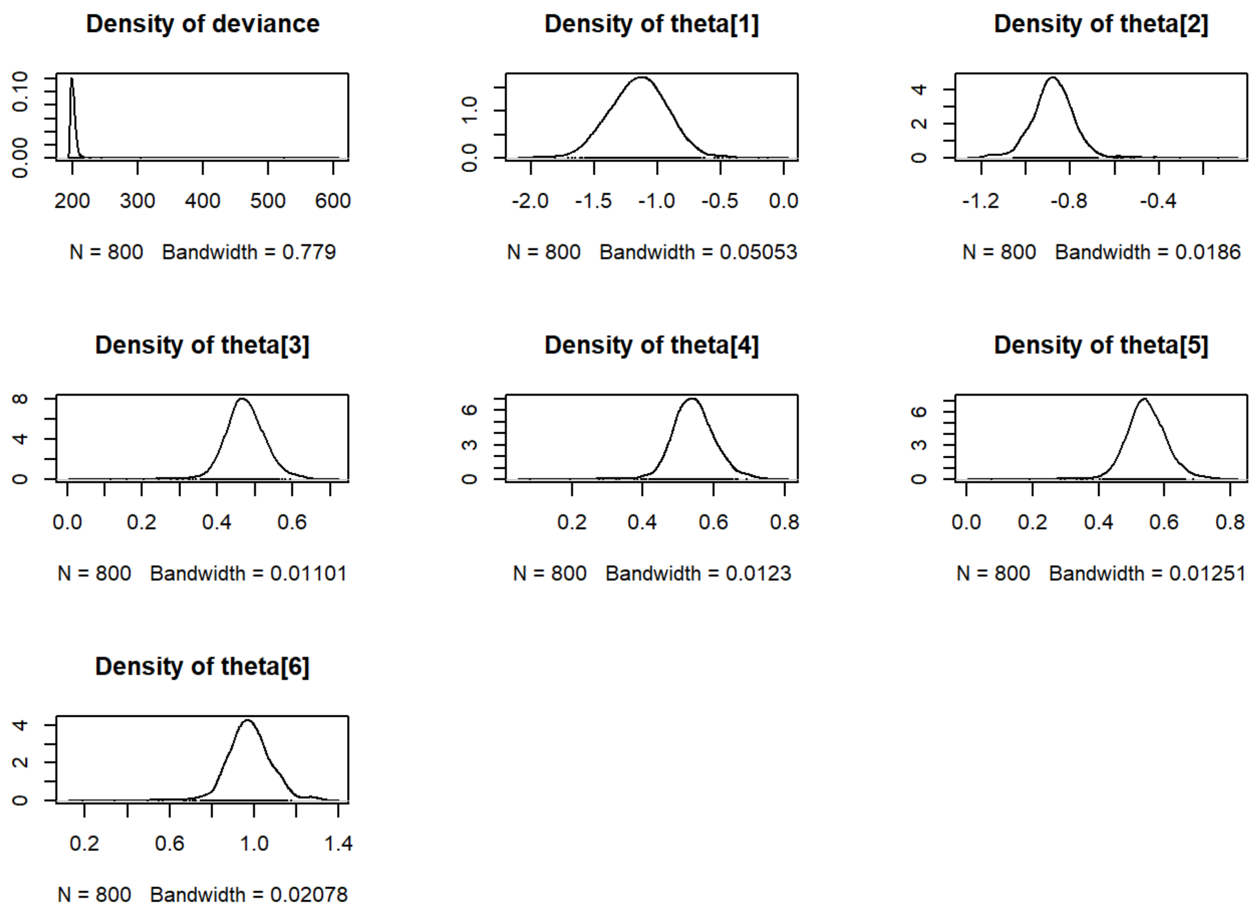

Trace of theta[3]
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
2000
4000
6000
8000
Iterations


Trace of theta[4]
0.1
0.2
0.3
0.4
0.5
0.6
0.7
2000
4000
6000
8000
Iterations


Trace of theta[5]
0.0
0.2
0.4
0.6
0.8
2000
4000
6000
8000
Iterations



```
# Density plots
plot(as.mcmc(modLogitSim.fit),density = T,trace = F)
```



```
#Gelmna-Rubin Test

gelman.diag(as.mcmc(modLogitSim.fit))
```

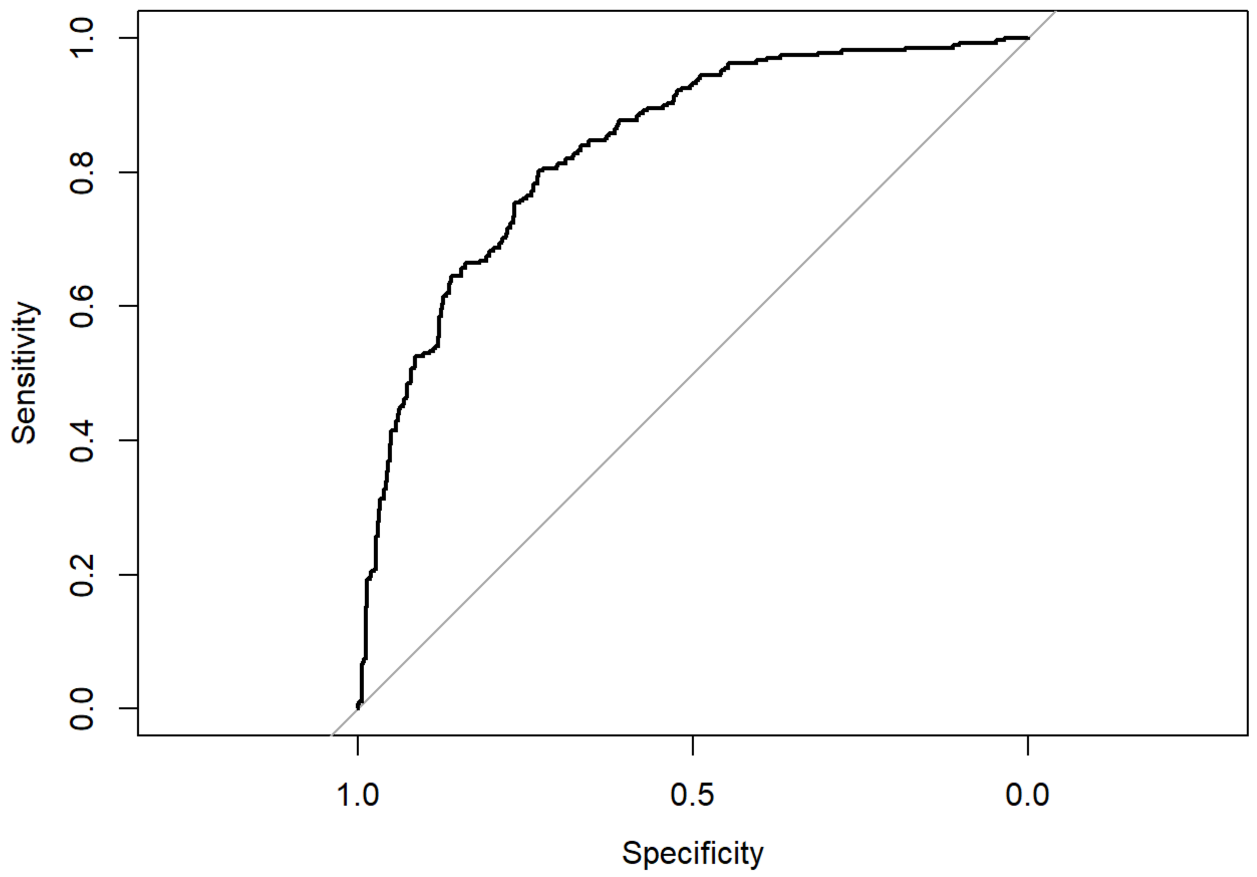
```
## Potential scale reduction factors:
##
##          Point est. Upper C.I.
## deviance       1.02       1.07
## theta[1]       1.04       1.14
## theta[2]       1.18       1.53
## theta[3]       1.15       1.44
## theta[4]       1.14       1.40
## theta[5]       1.17       1.49
## theta[6]       1.19       1.54
##
## Multivariate psrf
##
## 1.12
```

```
#gelman.plot(as.mcmc(modLogit.fit))

#Heidelberger-Welch Test
heidel.diag(as.mcmc(modLogitSim.fit))
```

```
## [[1]]
##
##          Stationarity start     p-value
##          test         iteration
## deviance passed       81        0.916
## theta[1] passed        1        0.354
## theta[2] passed        1        0.519
## theta[3] passed        1        0.311
## theta[4] passed        1        0.646
## theta[5] passed        1        0.434
## theta[6] passed        1        0.470
##
##          Halfwidth Mean    Halfwidth
##          test
## deviance passed    199.798 0.2809
## theta[1] passed     -1.120 0.0332
## theta[2] passed     -0.864 0.0270
## theta[3] passed      0.467 0.0142
## theta[4] passed      0.532 0.0142
## theta[5] passed      0.536 0.0155
## theta[6] passed      0.962 0.0268
```

```
##
## [[2]]
##
##          Stationarity start     p-value
##          test         iteration
## deviance passed       81         0.4043
## theta[1] passed        1         0.0527
## theta[2] passed        1         0.3680
## theta[3] passed        1         0.2412
## theta[4] passed        1         0.3185
## theta[5] passed        1         0.3101
## theta[6] passed        1         0.4817
##
##          Halfwidth Mean    Halfwidth
##          test
## deviance passed    200.180 0.6242
## theta[1] passed     -1.131 0.0561
## theta[2] passed     -0.854 0.0447
## theta[3] passed      0.462 0.0226
## theta[4] passed      0.532 0.0286
## theta[5] passed      0.530 0.0254
## theta[6] passed      0.953 0.0474
##
## [[3]]
##
##          Stationarity start     p-value
##          test         iteration
## deviance passed       1          0.206
## theta[1] passed       1          0.182
## theta[2] passed       1          0.398
## theta[3] passed       1          0.206
## theta[4] passed       1          0.290
## theta[5] passed       1          0.327
## theta[6] passed       1          0.221
##
##          Halfwidth Mean    Halfwidth
##          test
## deviance passed    201.568 1.7724
## theta[1] passed     -1.197 0.0480
## theta[2] passed     -0.909 0.0444
## theta[3] passed      0.490 0.0240
## theta[4] passed      0.561 0.0253
## theta[5] passed      0.563 0.0294
## theta[6] passed      1.010 0.0473
```

```
#ROC Curve of point estimates

X1=X[,1]
X2=X[,2]
X3=X[,3]
X4=X[,4]
X5=X[,5]



mcmc_fit_values_sim=theta.hat_sim.jags[-1][1]+theta.hat_sim.jags[-1][2]*X[,1]+theta.hat_sim.jags[-1][3]*X[,2]+theta.hat_sim.jags[-1][4]*X[,3]+theta.hat_sim.jags[-1][5]*X[,4]+theta.hat_sim.jags[-1][6]*X[,5]


roc(Y,exp(mcmc_fit_values_sim)/(exp(mcmc_fit_values_sim)+1),plot = T)
```



```
##
## Call:
## roc.default(response = Y, predictor = exp(mcmc_fit_values_sim)/(exp(mcmc_fit_values_sim) +     1), plot = T)
##
## Data: exp(mcmc_fit_values_sim)/(exp(mcmc_fit_values_sim) + 1) in 500 controls (Y 0) < 268 cases (Y 1).
## Area under the curve: 0.836
```

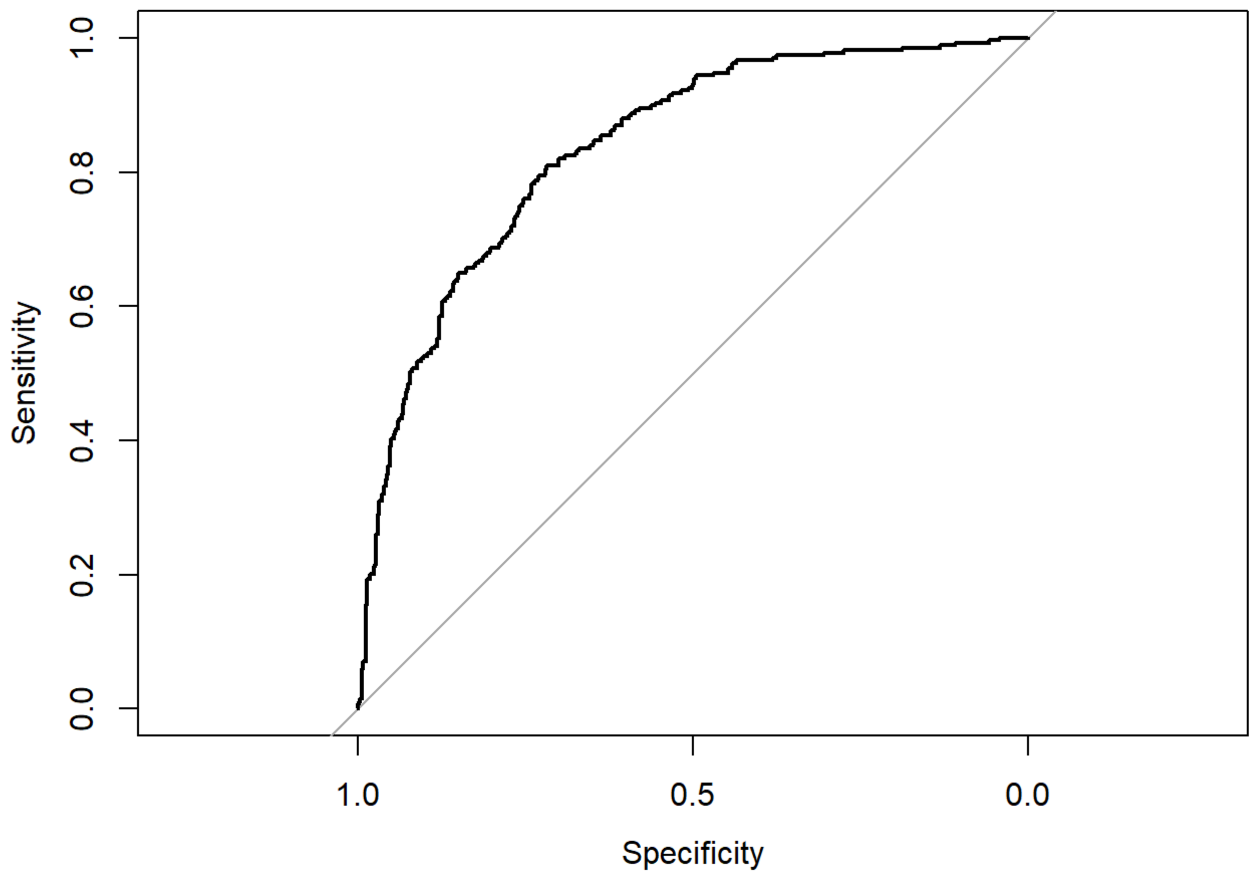
By running the MCMC with simulated data, we can notice that the model outputs similar point estimates and credible interval as the MCMC with the original data.Although the chains are not optimal and the convergence is not so obvious from the graphical illustrations,after performing the tests,we can conclude that the chains have converged and the model managed to recover its parameters and their intervals.We can also see the ROC curve of the point estimates, that has almost the same area under it for both original and simulated data.

Probit Model

We will propose a new model, similar to the previous one. The difference between a logit and a probit model is that logit uses a non-linear mapping(sigmoid) to values in the range of probabilities,ie[0,1].The probit model instead is written as $Pr(p|w) = \phi(X^T W)$, where Φ is the CDF of the standardized normal distribution. This again maps the ouptut into [0,1]. We will use the same Principal Components as in the first model as the variables of the Probit model.

Frequentist Approach For the MLE method of the probit model, we will be using directly the glm function, that comes in handy with the p-values and confidence intervals of the parameters,since we do not really want to reinvent the wheel.

```r
data=read.csv('D:\\sapienza\\SDS2\\diabetes.csv')


Y=ifelse(data$Class == "positive",1,0)

data$Class<-NULL

n<-length(Y)
#data$Class<-Y

pca=prcomp(data,center = T,scale. = T)


X= pca$x


X=select(as.data.frame(X),PC1,PC2,PC3,PC5,PC6)



lr=glm(Y~as.matrix(X),family = binomial(probit))


roc(Y,lr$fitted.values,plot=T)
```



```
##
## Call:
## roc.default(response = Y, predictor = lr$fitted.values, plot = T)
##
## Data: lr$fitted.values in 500 controls (Y 0) < 268 cases (Y 1).
## Area under the curve: 0.836
```

```
summary(lr)
```

```
##
## Call:
## glm(formula = Y ~ as.matrix(X), family = binomial(probit))
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7075  -0.7401  -0.4107   0.8165   2.8222
##
## Coefficients:
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -0.52053    0.05471  -9.515  < 2e-16 ***
## as.matrix(X)PC1 -0.44975    0.04210 -10.684  < 2e-16 ***
## as.matrix(X)PC2  0.25414    0.03949   6.435 1.23e-10 ***
## as.matrix(X)PC3  0.26683    0.05448   4.897 9.71e-07 ***
## as.matrix(X)PC5  0.25896    0.06095   4.249 2.15e-05 ***
## as.matrix(X)PC6  0.49296    0.06933   7.111 1.16e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 993.48  on 767  degrees of freedom
## Residual deviance: 733.85  on 762  degrees of freedom
## AIC: 745.85
##
## Number of Fisher Scoring iterations: 5
```

```
confint.default(lr)
```

```
##                      2.5 %     97.5 %
## (Intercept)     -0.6277624 -0.4133052
## as.matrix(X)PC1 -0.5322556 -0.3672441
## as.matrix(X)PC2  0.1767351  0.3315477
## as.matrix(X)PC3  0.1600442  0.3736154
## as.matrix(X)PC5  0.1394935  0.3784203
## as.matrix(X)PC6  0.3570814  0.6288404
```

We can see that the MLE of the probit model returns a similar ROC curve, with the same area under it as with the logit model.

MCMC

```
require(R2jags)
require(statip)
require(pROC)
# MCMC --------------------------------------------------------------------
modelProbit <- function() {
  # Likelihood
  for(i in 1:n)
  {
    y[i] ~ dbern(p[i])

    probit(p[i]) <- theta[1] + theta[2]*X1[i] +theta[3]*X2[i]+theta[4]*X3[i]+theta[5]*X4[i]+theta[6]*X5[i]
  }

  # Priors
  theta[1] ~ dnorm(0, 1.0E-6)
  for (i in 1:n_param+1)
  {
    theta[i] ~ dnorm(0, 1.0E-6)

  }
}

# Preparing data for JAGS -------------------------------------------------
X1=X[,1]
X2=X[,2]
X3=X[,3]
X4=X[,4]
X5=X[,5]
y=Y
n=nrow(X)
```

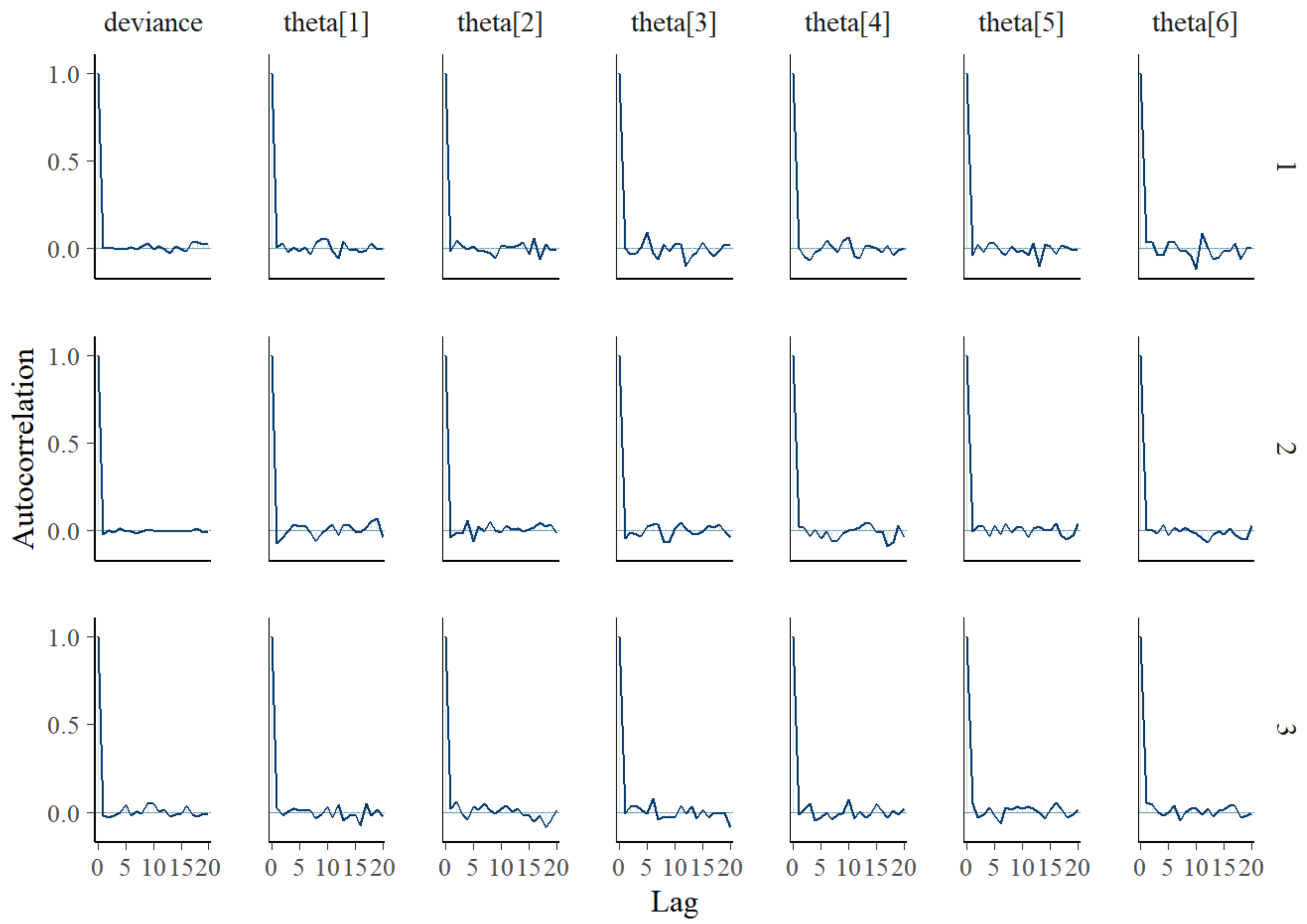
```
n_param=ncol(X)

dat.jags <- list("X1", "X2", "X3", "X4", "X5", "y", "n","n_param")



# Parameters ---------------------------------------------------------------

modProbit.params="theta"

# Starting values
modProbit.inits <- function(){
  list("theta" = rnorm(n_param+1)
  )
}


# Run JAGS -----------------------------------------------------------------

set.seed(123)
modProbit.fit <- jags(data = dat.jags,                                    # DATA
                      model.file = modelProbit, inits = modProbit.inits,          # MODEL
                      parameters.to.save = modProbit.params,
                      n.chains = 3, n.iter = 9000, n.burnin = 1000, n.thin=10)      # MCMC
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 768
##    Unobserved stochastic nodes: 6
##    Total graph size: 9994
##
## Initializing model
```

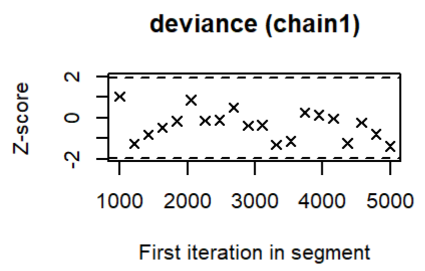
```
modProbit.fit
```

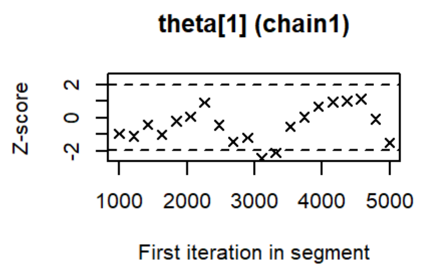
```
## Inference for Bugs model at "C:/Users/teoso/AppData/Local/Temp/RtmpCKRd8h/model31c893d202f.txt", fit using jag
s,
##  3 chains, each with 9000 iterations (first 1000 discarded), n.thin = 10
##  n.sims = 2400 iterations saved
##          mu.vect sd.vect    2.5%     25%     50%     75%   97.5%  Rhat
## theta[1]  -0.522   0.055  -0.629  -0.560  -0.521  -0.484  -0.419 1.001
## theta[2]  -0.452   0.041  -0.530  -0.479  -0.453  -0.426  -0.374 1.001
## theta[3]   0.256   0.042   0.179   0.230   0.255   0.281   0.337 1.001
## theta[4]   0.268   0.053   0.166   0.234   0.268   0.302   0.369 1.001
## theta[5]   0.260   0.059   0.145   0.221   0.260   0.300   0.375 1.001
## theta[6]   0.496   0.073   0.363   0.447   0.495   0.542   0.640 1.006
## deviance 740.044   7.809 735.085 737.300 739.223 741.604 748.287 1.055
##          n.eff
## theta[1]  1900
## theta[2]  2400
## theta[3]  2400
## theta[4]  2000
## theta[5]  2400
## theta[6]   390
## deviance  2400
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 30.5 and DIC = 770.6
## DIC is an estimate of expected predictive error (lower deviance is better).
```

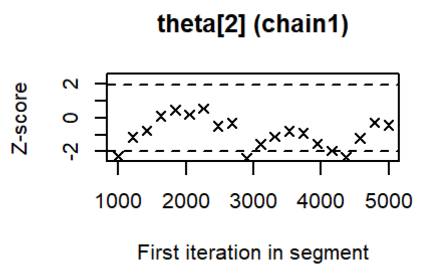
```
chainArray <- modProbit.fit$BUGSoutput$sims.array


# Plots with BayesPlot
#bayesplot::mcmc_combo(chainArray)
bayesplot::mcmc_acf(chainArray)
```

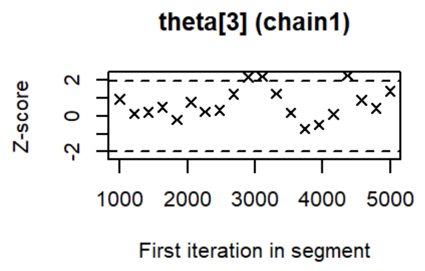


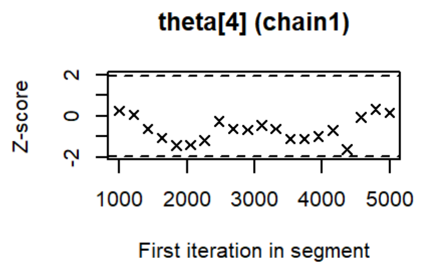
```
# Diagnostic with coda
coda.fit <- as.mcmc(modProbit.fit)

coda::geweke.plot(coda.fit)
```

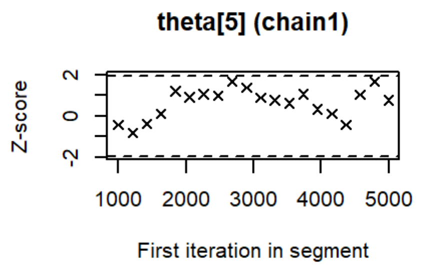


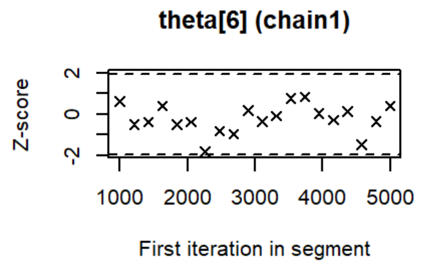


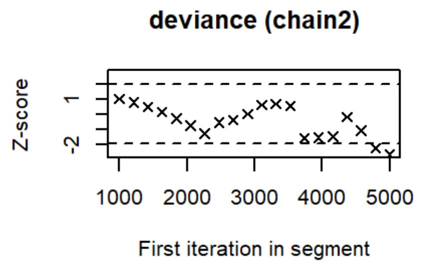


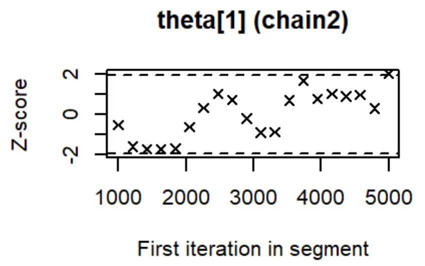


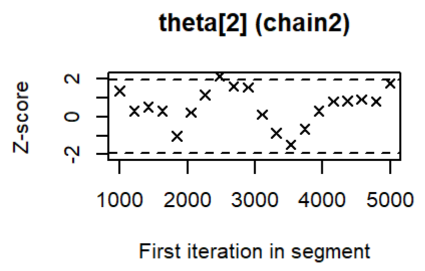


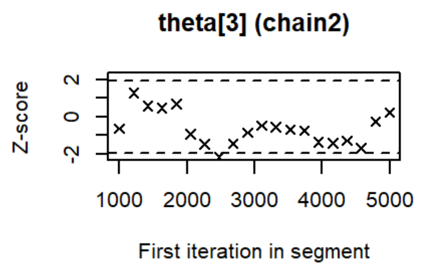


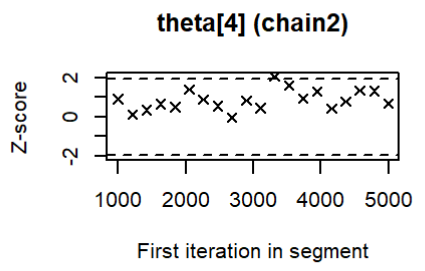




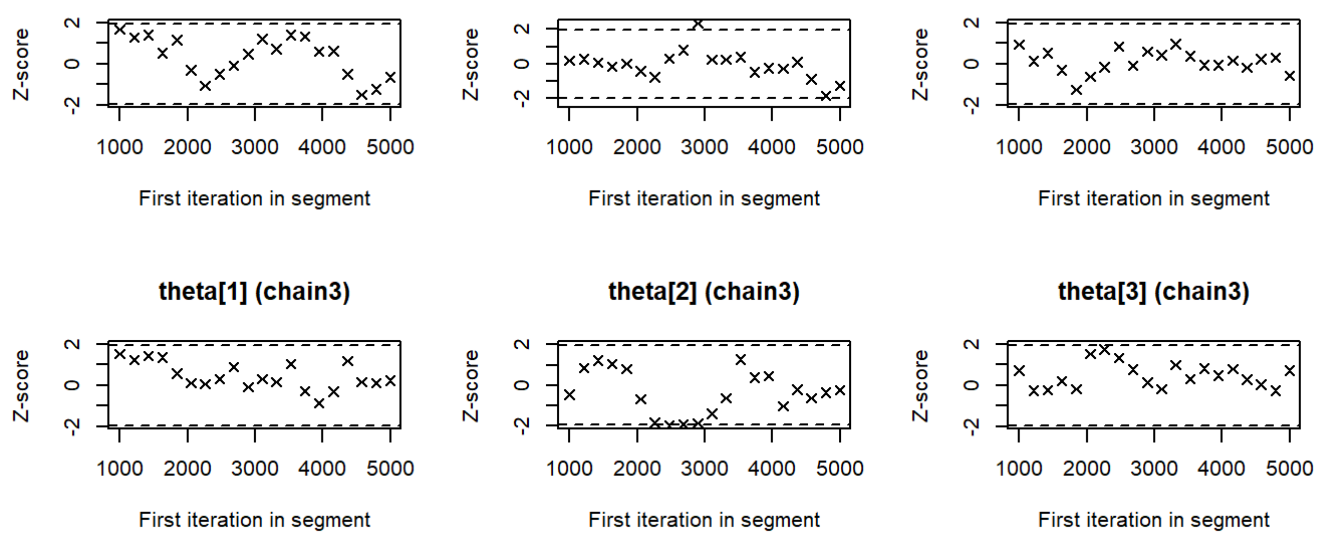


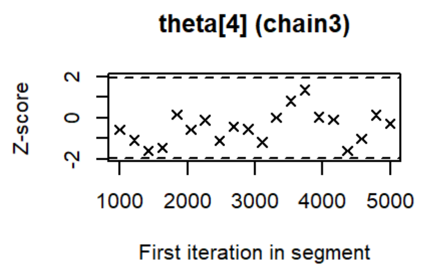


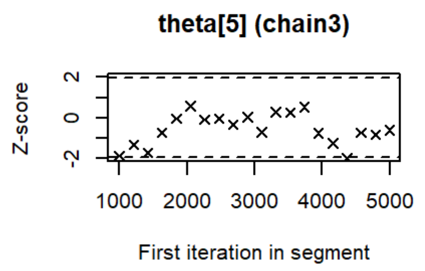


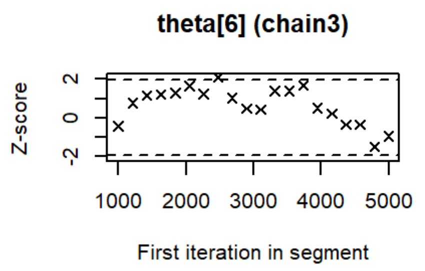


theta[5] (chain2) **theta[6] (chain2)** **deviance (chain3)**

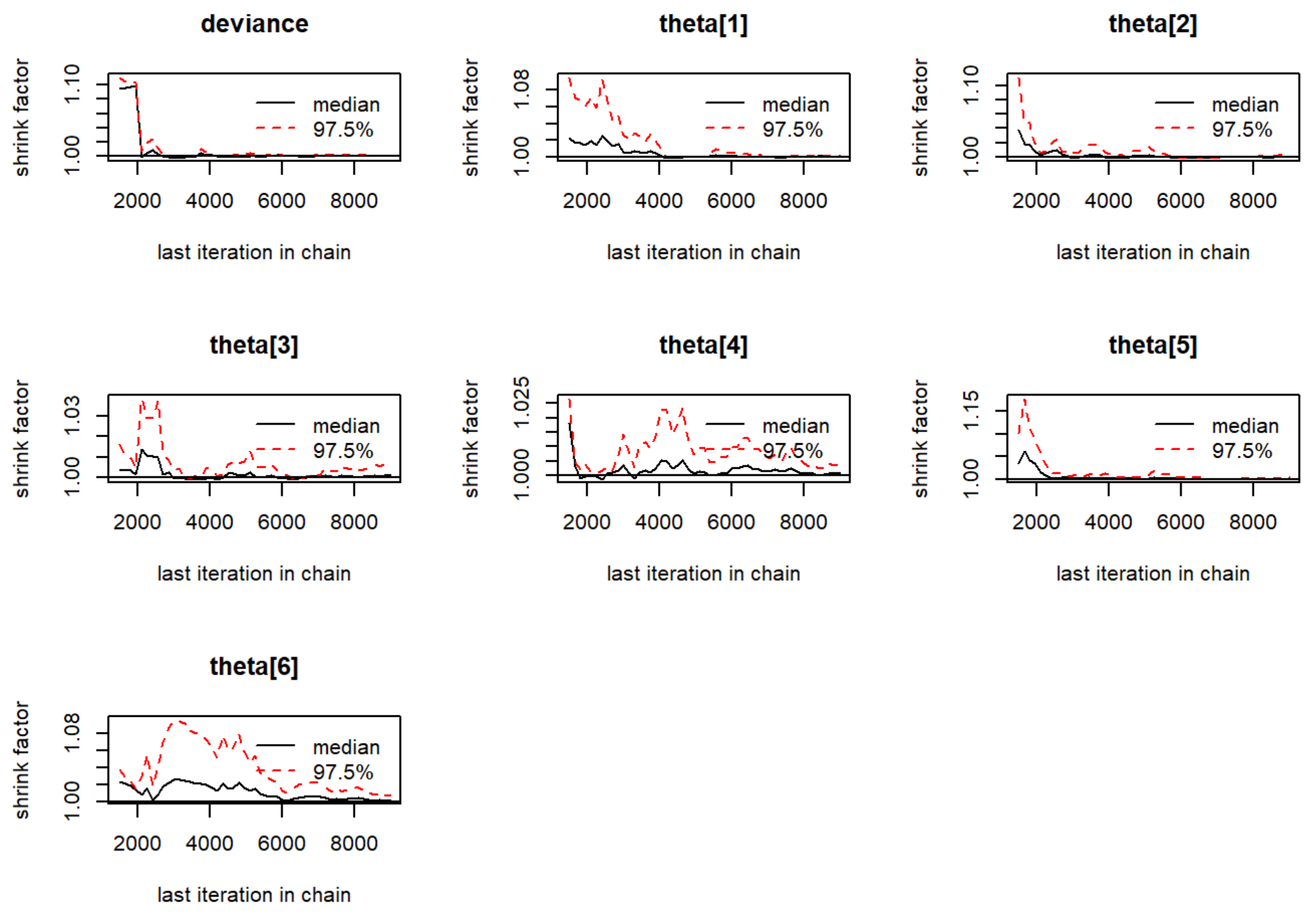

Z-score
First iteration in segment
1000 2000 3000 4000 5000
-2 0 2
theta[1] (chain3)
theta[2] (chain3)
theta[3] (chain3)







```
coda::gelman.plot(coda.fit)
```



```
# we can see that the chains are good,although perhaps not optimal.


# Manipulating the chain -----------------------------------------------------------

chainMat <- modProbit.fit$BUGSoutput$sims.matrix

# Point estimates
(theta.hat.jags <- colMeans(chainMat))
```

```
##    deviance    theta[1]    theta[2]    theta[3]    theta[4]    theta[5]
## 740.0443649  -0.5217235  -0.4524290   0.2557826   0.2678606   0.2603645
##    theta[6]
##   0.4959219
```

```
# Intervals
cred <- 0.95
(theta.ET.jags <- apply(chainMat, 2, quantile, prob=c((1-cred)/2, 1-(1-cred)/2)))
```

```
##       deviance   theta[1]   theta[2]  theta[3]  theta[4]  theta[5]
## 2.5%  735.0845 -0.6294122 -0.5296796 0.1786064 0.1662596 0.1449910
## 97.5% 748.2867 -0.4187642 -0.3744101 0.3367666 0.3686188 0.3745723
##        theta[6]
## 2.5%  0.3630632
## 97.5% 0.6404717
```

```
# What about the HPD?
(theta.HPD.jags <- coda::HPDinterval(as.mcmc(chainMat)))
```

```
##                lower       upper
## deviance 734.5108071 746.6759552
## theta[1]  -0.6273616  -0.4169014
## theta[2]  -0.5299321  -0.3746671
## theta[3]   0.1764091   0.3351274
## theta[4]   0.1629417   0.3652562
## theta[5]   0.1450101   0.3746331
## theta[6]   0.3599095   0.6363079
## attr(,"Probability")
## [1] 0.95
```

```
cat('parameters via MCMC with original data are', theta.hat.jags[-1], 'parameters via MLE are:',lr$coefficients)
```

```
## parameters via MCMC with original data are -0.5217235 -0.452429 0.2557826 0.2678606 0.2603645 0.4959219 parame
ters via MLE are: -0.5205338 -0.4497499 0.2541414 0.2668298 0.2589569 0.4929609
```

```
# We can observe that the point estimates of the parameters with the MCMC are close to the ones from the MLE.

#Error approximation
errors=sapply(modProbit.fit$BUGSoutput$sd, function(x) x^2/(modProbit.fit$n.iter-1000))
cat('approximation errors are ',errors$theta)
```

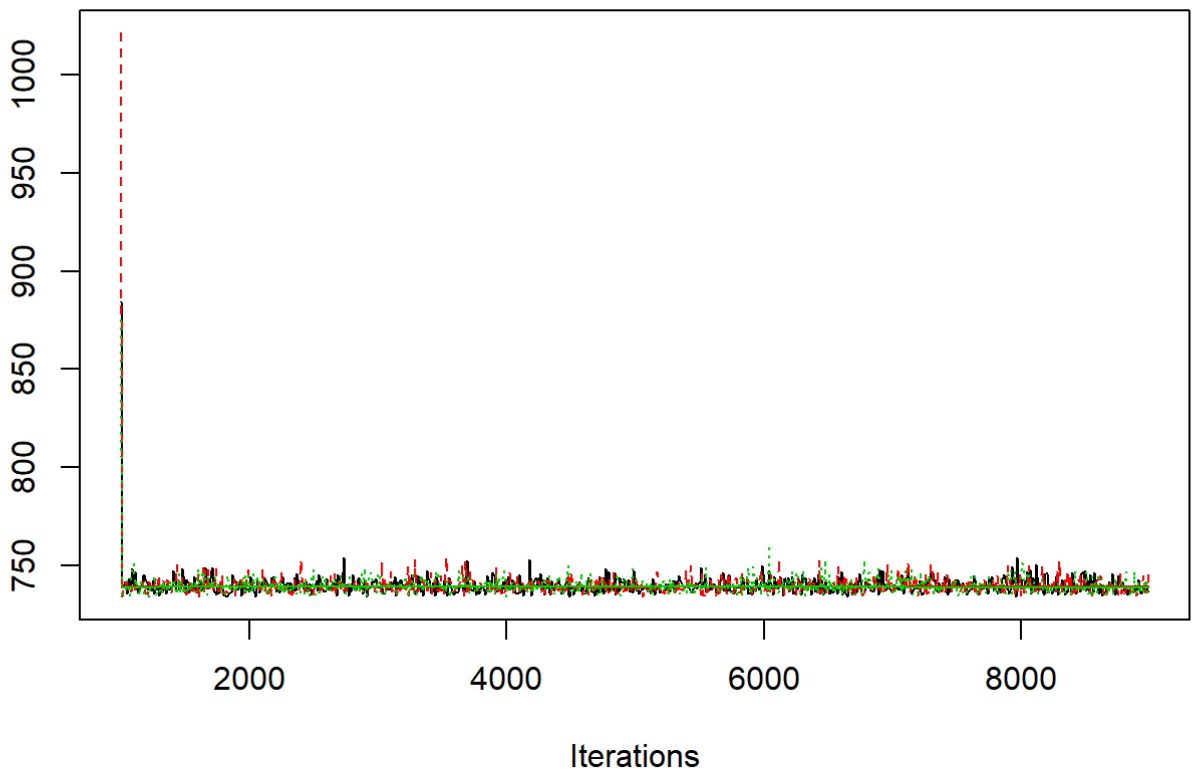
```
## approximation errors are  3.80381e-07 2.06897e-07 2.182217e-07 3.467533e-07 4.35711e-07 6.688273e-07
```

```
# Parameter uncertainty
cat('uncertainty for parameters',abs(modProbit.fit$BUGSoutput$sd$theta/theta.hat.jags[-1]))
```

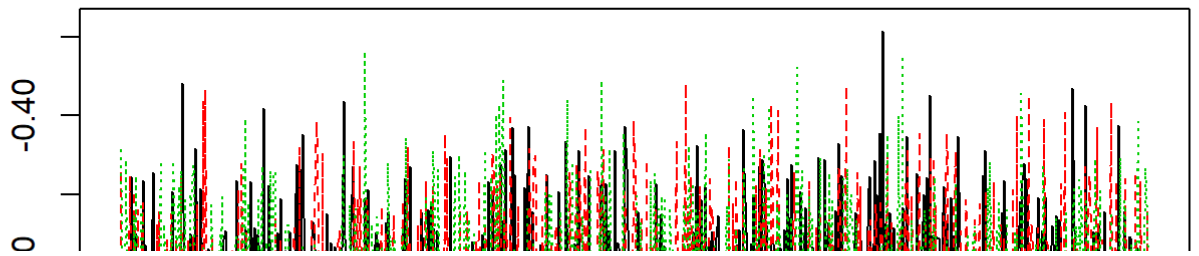
```
## uncertainty for parameters 0.1057338 0.08992319 0.1633514 0.1966285 0.2267579 0.1474989
```

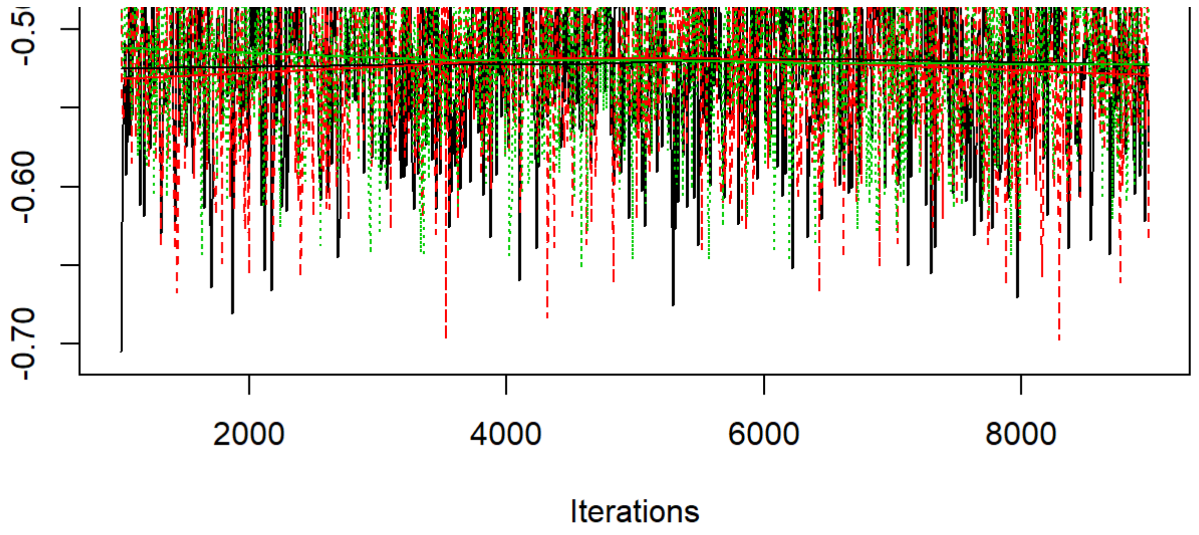
```
#Traceplot
traceplot(as.mcmc(modProbit.fit))
```

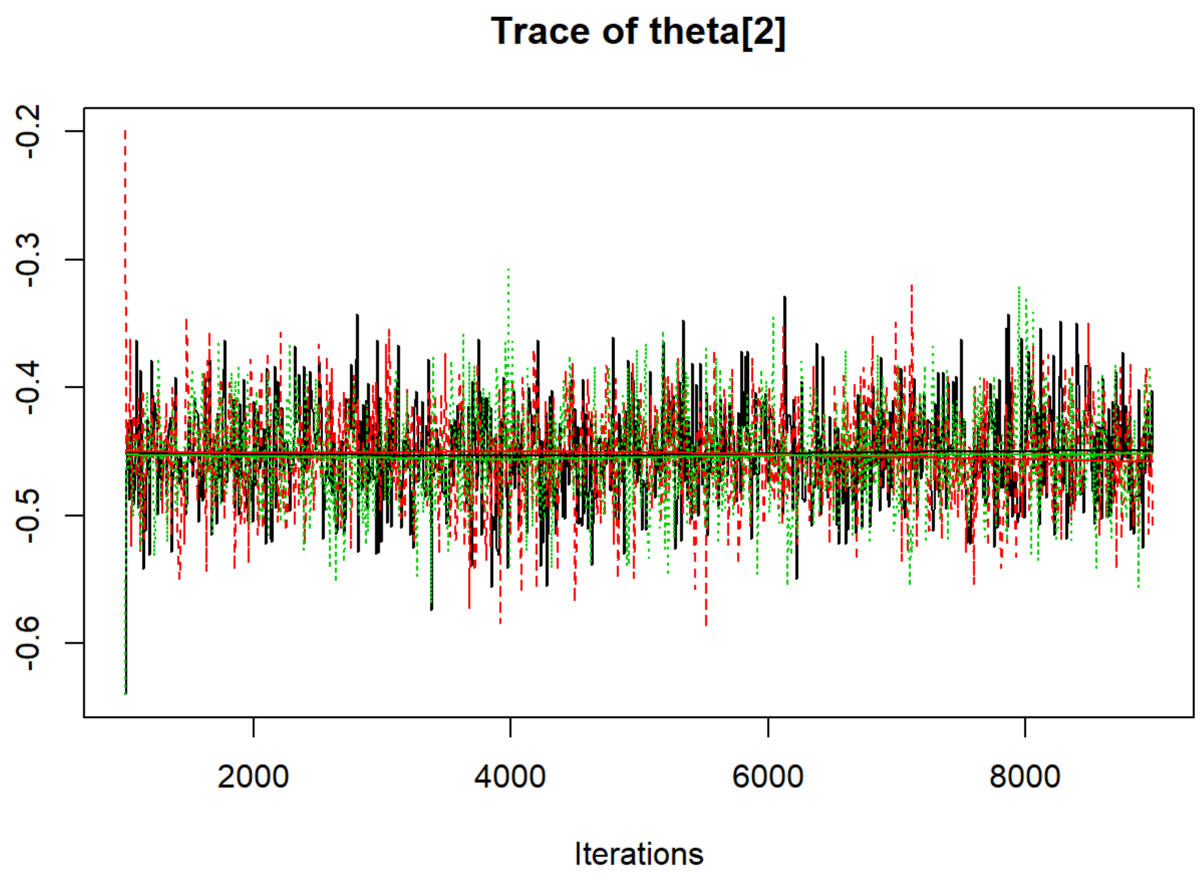
Trace of deviance



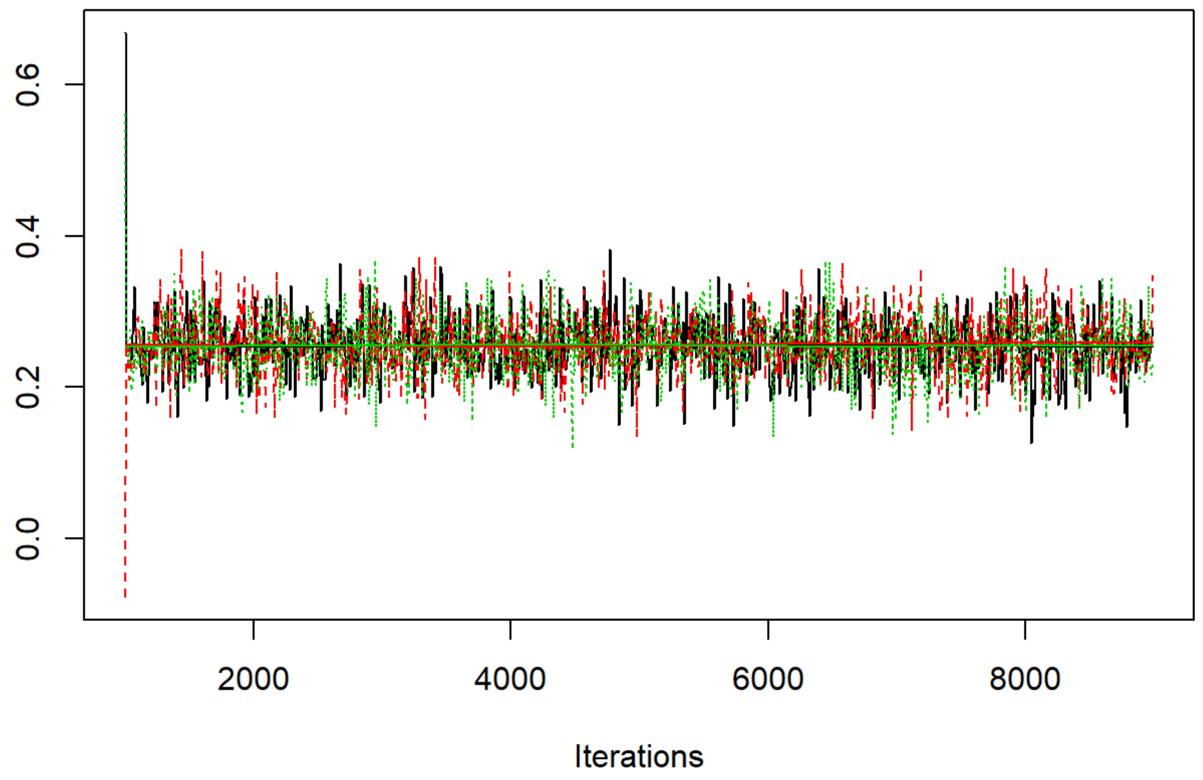
Trace of theta[1]

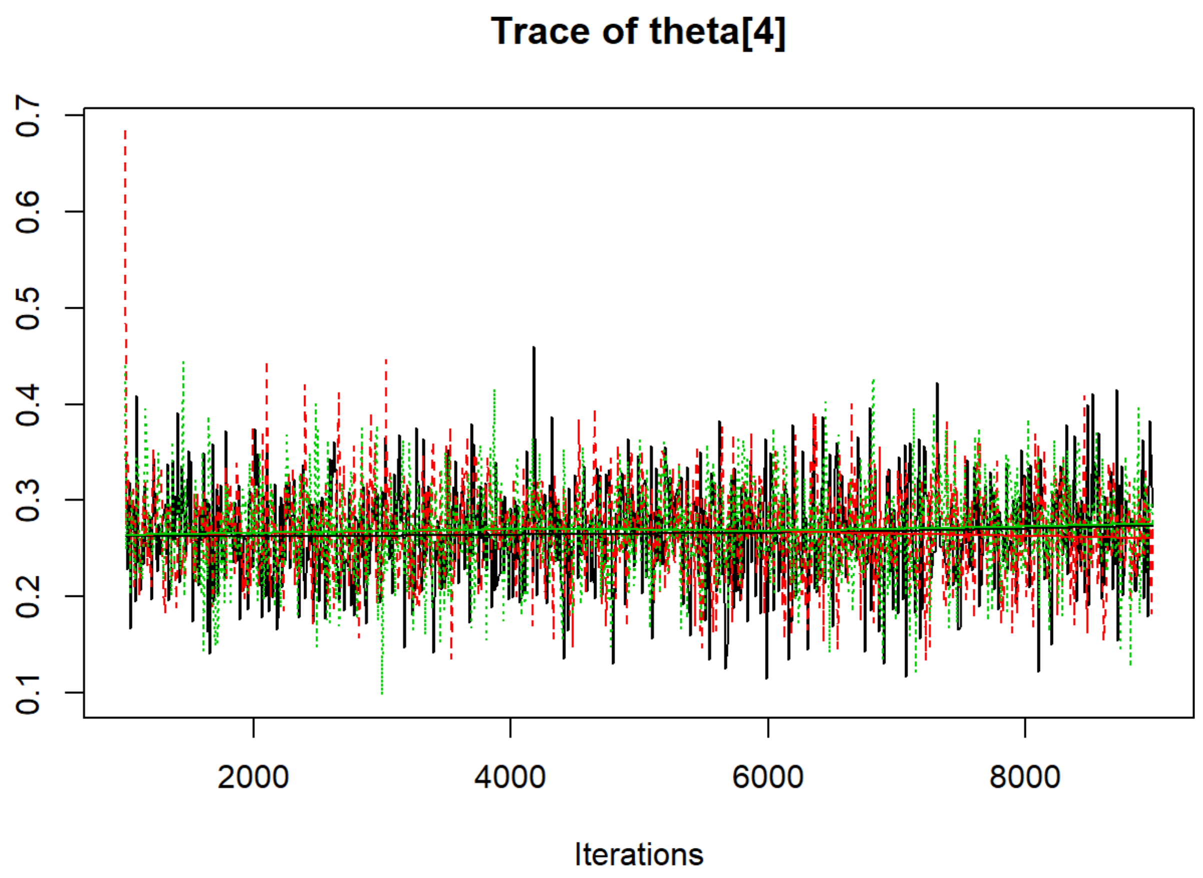


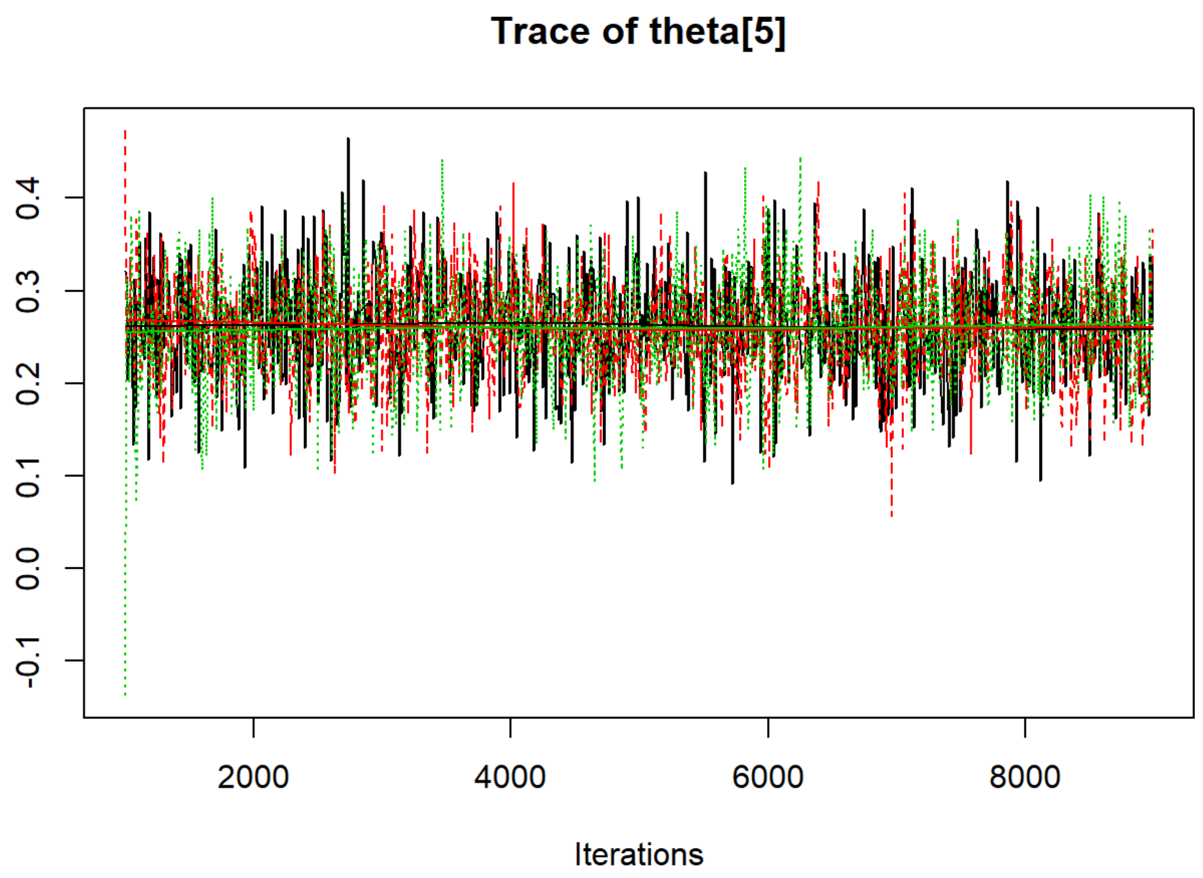

-0.5
-0.60
-0.70
2000
4000
6000
8000
Iterations

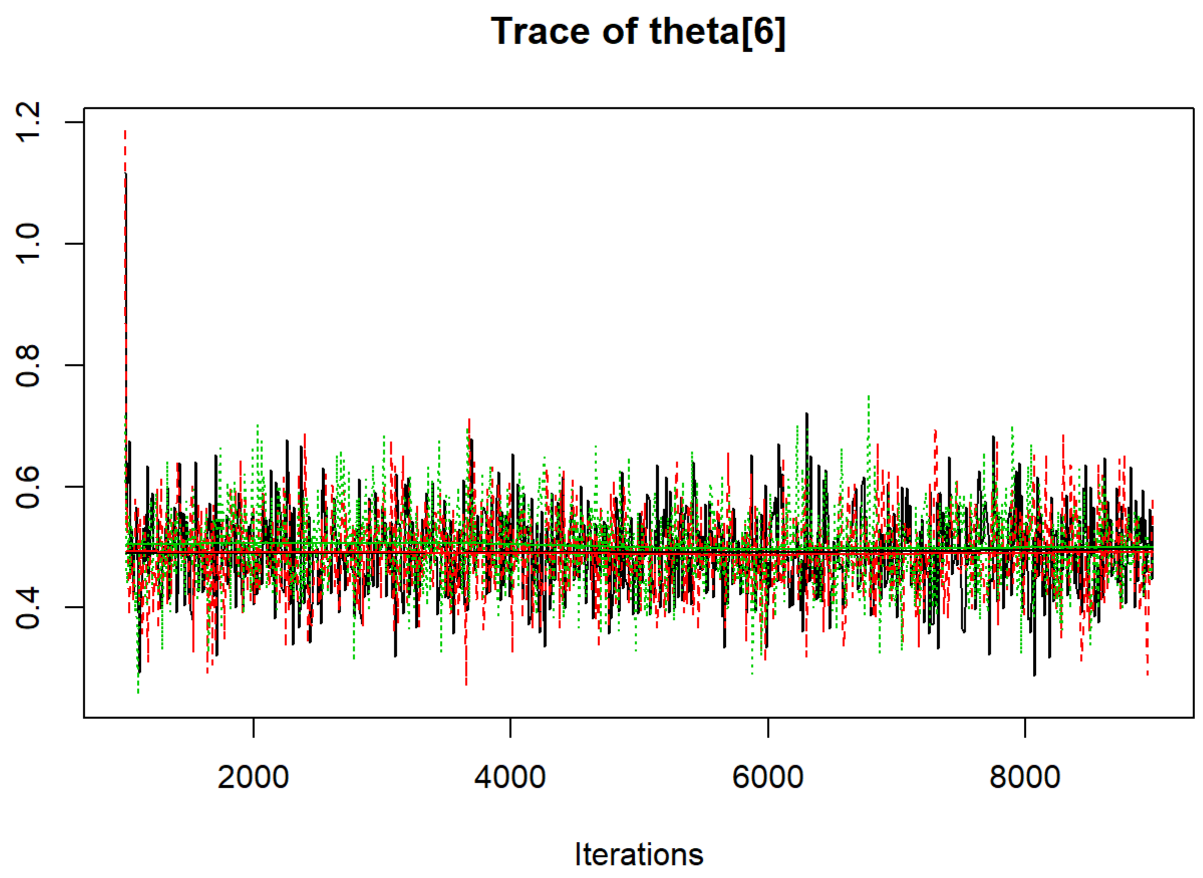

Trace of theta[2]
-0.2
-0.3
-0.4
-0.5
-0.6
2000
4000
6000
8000
Iterations

Trace of theta[3]

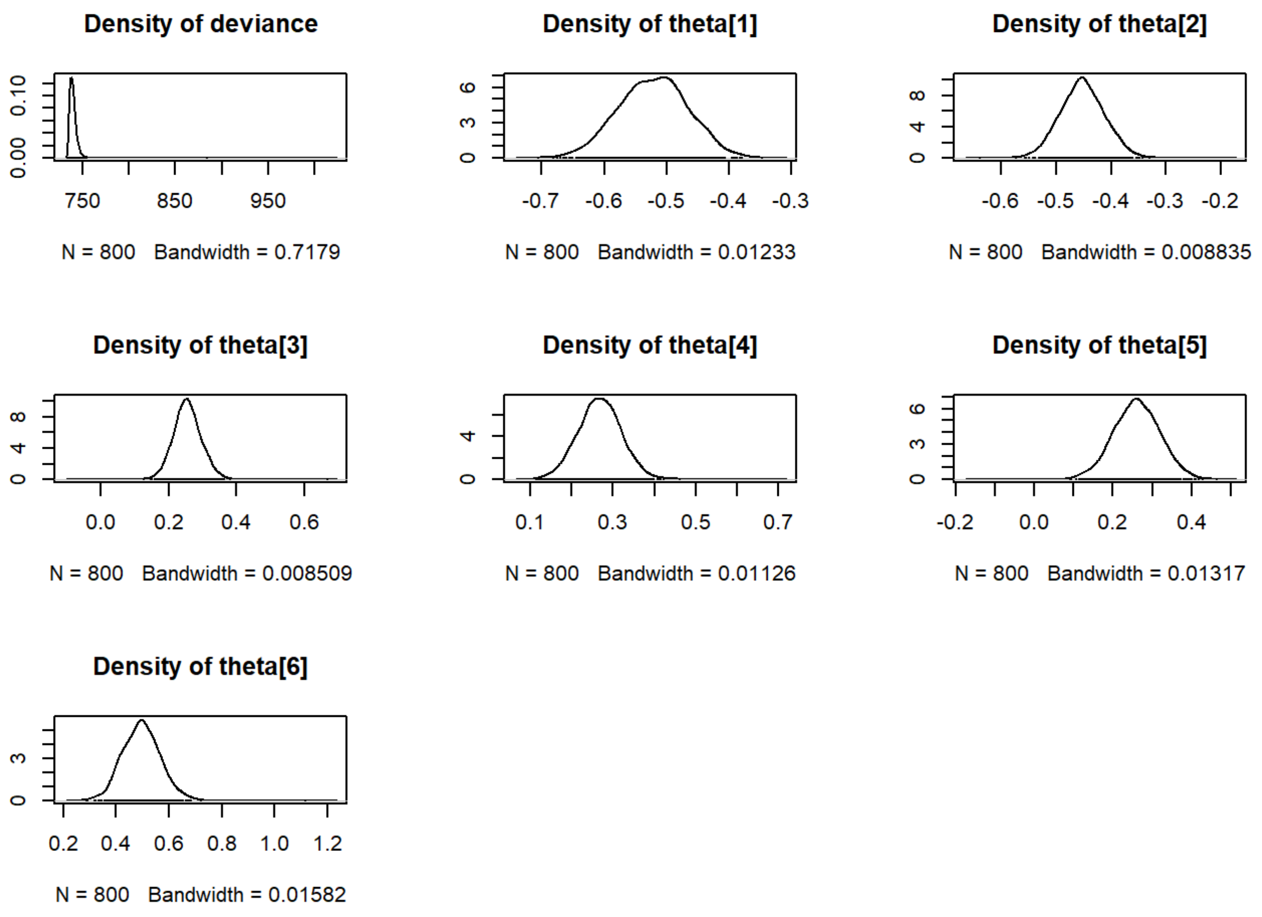



Trace of theta[4]
0.1
0.2
0.3
0.4
0.5
0.6
0.7
2000
4000
6000
8000
Iterations


Trace of theta[5]
0.4
0.3
0.2
0.1
0.0
-0.1
2000
4000
6000
8000
Iterations



```
# Density plots
plot(as.mcmc(modProbit.fit),density = T,trace = F)
```



```
#Gelmna-Rubin Test

gelman.diag(as.mcmc(modProbit.fit))
```

```
## Potential scale reduction factors:
##
##          Point est. Upper C.I.
## deviance          1       1.00
## theta[1]          1       1.00
## theta[2]          1       1.01
## theta[3]          1       1.01
## theta[4]          1       1.00
## theta[5]          1       1.00
## theta[6]          1       1.01
##
## Multivariate psrf
##
## 1.01
```

```
#gelman.plot(as.mcmc(modLogit.fit))

#Heidelberger-Welch Test
heidel.diag(as.mcmc(modProbit.fit))
```

```
## [[1]]
##
##          Stationarity start     p-value
##          test         iteration
## deviance passed       81        0.530
## theta[1] passed        1        0.434
## theta[2] passed        1        0.181
## theta[3] passed        1        0.197
## theta[4] passed        1        0.275
## theta[5] passed        1        0.393
## theta[6] passed        1        0.857
##
##          Halfwidth Mean    Halfwidth
##          test
## deviance passed    739.784 0.24038
## theta[1] passed     -0.522 0.00379
## theta[2] passed     -0.451 0.00277
## theta[3] passed      0.255 0.00292
## theta[4] passed      0.266 0.00368
## theta[5] passed      0.262 0.00414
## theta[6] passed      0.494 0.00458
```

```
##
## [[2]]
##
##          Stationarity start     p-value
##          test         iteration
## deviance passed       81         0.604
## theta[1] passed        1         0.298
## theta[2] passed        1         0.276
## theta[3] passed        1         0.353
## theta[4] passed        1         0.151
## theta[5] passed        1         0.212
## theta[6] passed        1         0.904
##
##          Halfwidth Mean    Halfwidth
##          test
## deviance passed    739.838 0.24083
## theta[1] passed     -0.523 0.00368
## theta[2] passed     -0.452 0.00291
## theta[3] passed      0.257 0.00289
## theta[4] passed      0.268 0.00365
## theta[5] passed      0.260 0.00394
## theta[6] passed      0.491 0.00520
##
## [[3]]
##
##          Stationarity start     p-value
##          test         iteration
## deviance passed       81         0.565
## theta[1] passed        1         0.359
## theta[2] passed        1         0.460
## theta[3] passed        1         0.404
## theta[4] passed        1         0.507
## theta[5] passed        1         0.106
## theta[6] passed        1         0.205
##
##          Halfwidth Mean    Halfwidth
##          test
## deviance passed    739.787 0.25083
## theta[1] passed     -0.519 0.00371
## theta[2] passed     -0.453 0.00278
## theta[3] passed      0.255 0.00288
## theta[4] passed      0.270 0.00362
## theta[5] passed      0.259 0.00441
## theta[6] passed      0.503 0.00523
```

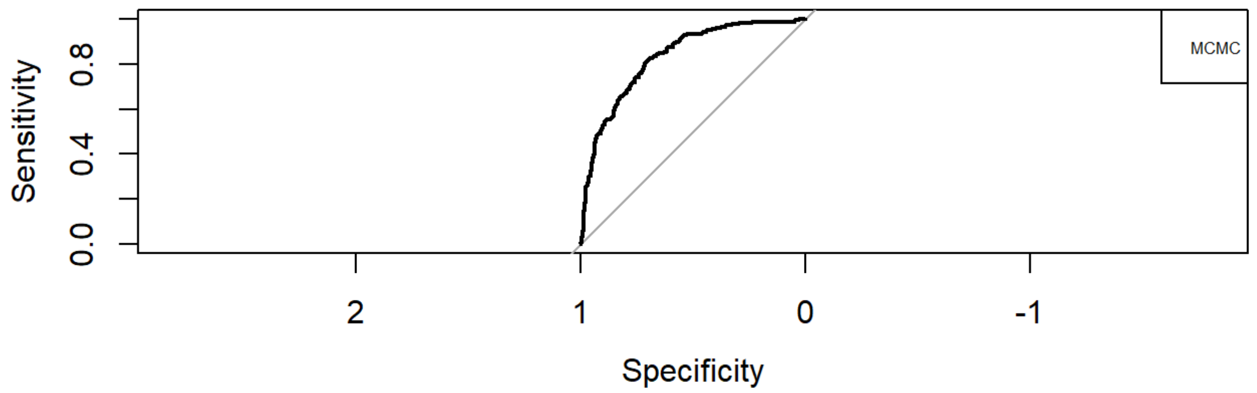
```
# ROC CURVES OF POINT ESTIMATES



par(mfrow=c(2,1))
fit_values_MC=theta.hat.jags[-1][1]+theta.hat.jags[-1][2]*X[,1]+theta.hat.jags[-1][3]*X[,2]+theta.hat.jags[-1][4]
*X[,3]+theta.hat.jags[-1][5]*X[,4]+theta.hat.jags[6]*X[,5]
roc(Y,fit_values_MC,plot=T)
```

```
##
## Call:
## roc.default(response = Y, predictor = fit_values_MC, plot = T)
##
## Data: fit_values_MC in 500 controls (Y 0) < 268 cases (Y 1).
## Area under the curve: 0.8315
```

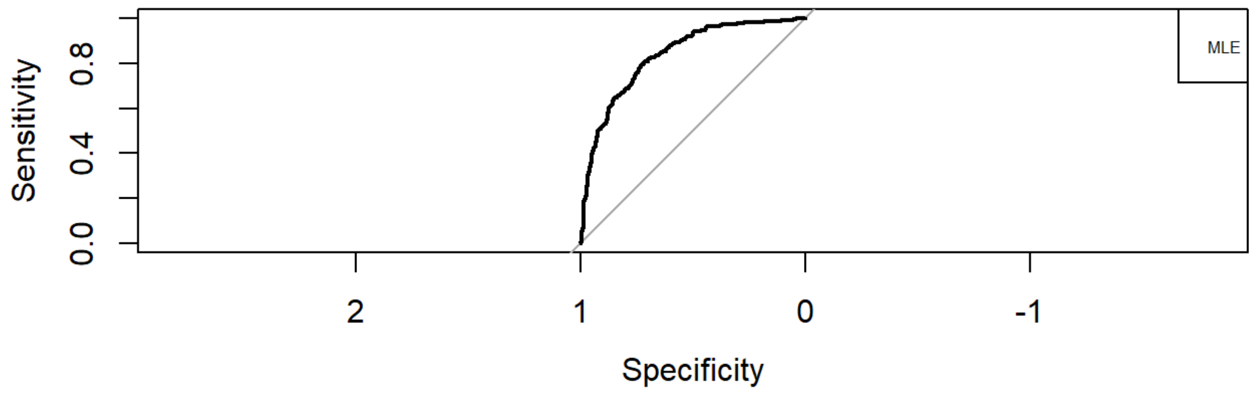
```
legend('topright',cex = 0.5,legend = 'MCMC')
roc(Y,lr$fitted.values,plot=T)
```

```
##
## Call:
## roc.default(response = Y, predictor = lr$fitted.values, plot = T)
##
## Data: lr$fitted.values in 500 controls (Y 0) < 268 cases (Y 1).
## Area under the curve: 0.836
```

```
legend('topright',cex = 0.5,legend = 'MLE')
```



We see that our chains successfully



converged to the stationary distribution. The point estimates and the intervals are identical with the ones we obtained with the MLE method,something that is depicted also in the ROC curves. Now, what remains to be done is to check the ability of the probit model to recover its true parameters, when the data are simulated. If it succeeds in it, we will be comparing our 2 models, in terms of DIC.

Simulate Data from Model hypothesis

```
# MCMC SIMULATED DATA ----------------------------------------------------
```

```
N <- 1000

X1 <- rnorm(N,0,8)
X2 <- rnorm(N,0,8)
X3 <- rnorm(N,0,8)
X4 <- rnorm(N,0,8)
X5 <- rnorm(N,0,8)
# Since our variables are the scaled PCA components, the standard deviation of 8 is quite big and could include o
utliers.



# Pick fixed values for the parameters of the model
theta=c(-0.5217235, -0.452429, 0.2557826, 0.2678606, 0.2603645, 0.4959219)

# Simulate response according to the model
linpred <- theta[1] + theta[2]*X1 +theta[3]*X2+theta[4]*X3+theta[5]*X4+theta[6]*X5
pis <- pnorm(linpred)

y=replicate(N,NA)
for(i in 1:N){
  y[i]=rbern(1,pis[i])
}
#y<-rbern(N,pis)

dat <- data.frame(X1=X1, X2=X2,X3=X3,X4=X4,X5=X5,y=y)




# MCMC inference -----------------------------------------------------------




library(R2jags)


modelProbitSim <- function() {
  # Likelihood
  for(i in 1:n)
  {
    y[i] ~ dbern(p[i])

    probit(p[i]) <- theta[1] + theta[2]*X1[i] +theta[3]*X2[i]+theta[4]*X3[i]+theta[5]*X4[i]+theta[6]*X5[i]


  }


  # Priors
  theta[1] ~ dnorm(0, 1.0E-6)
  for (i in 2:n_param)
  {
    theta[i] ~ dnorm(0, 1.0E-6)

  }
}

# Preparing data for JAGS -------------------------------------------------
X1=dat$X1
X2=dat$X2
X3=dat$X3
X4=dat$X4
X5=dat$X5
y=dat$y
n=nrow(dat)
n_param=ncol(dat)

dat.jags <- list("X1", "X2", "X3", "X4", "X5", "y", "n","n_param")


# Parameters ---------------------------------------------------------------

modProbitSim.params=c("theta")
```

```
# Starting values
modProbitSim.inits <- function(){
  list("theta" = c(rnorm(n_param, 0, 0.1))
  )
}


set.seed(123)
modProbitSim.fit <- jags(data = dat.jags,                                    # DATA
                         model.file = modelProbitSim, inits = modProbitSim.inits,          # MODEL
                         parameters.to.save = modProbitSim.params,
                         n.chains = 4, n.iter = 9000, n.burnin = 1000, n.thin=10)      # MCMC
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1000
##    Unobserved stochastic nodes: 6
##    Total graph size: 13010
##
## Initializing model
```

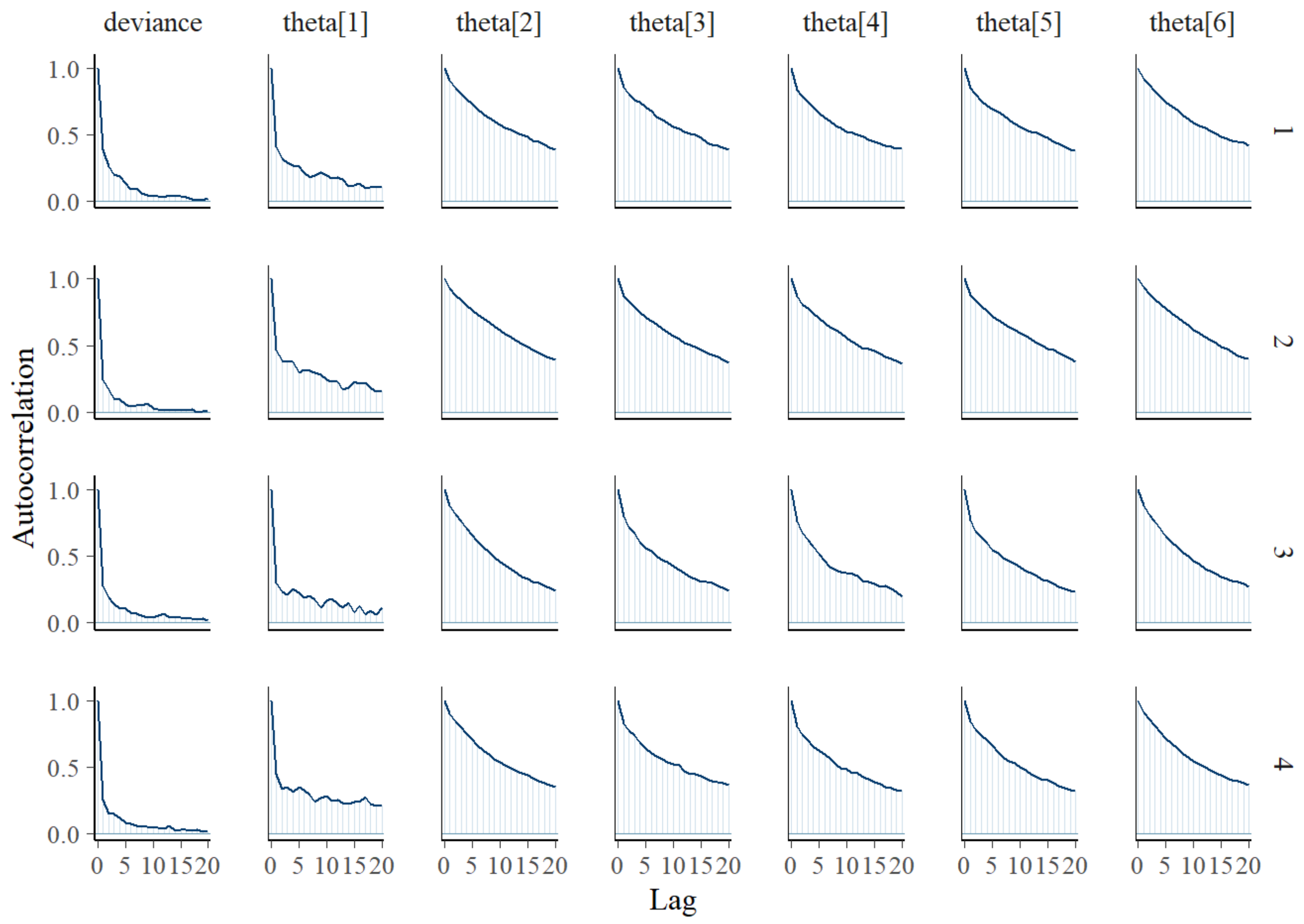
```
# for 3 chains,9000,1000,20 seems to be the closest we can get to the stationary distribution


# Results and diagnostics -------------------------------------------------

modProbitSim.fit
```

```
## Inference for Bugs model at "C:/Users/teoso/AppData/Local/Temp/RtmpCKRd8h/model31c81c14b59.txt", fit using jag
s,
##  4 chains, each with 9000 iterations (first 1000 discarded), n.thin = 10
##  n.sims = 3200 iterations saved
##          mu.vect sd.vect    2.5%     25%     50%     75%   97.5%  Rhat
## theta[1]  -0.737   0.141  -1.026  -0.827  -0.735  -0.647  -0.470 1.030
## theta[2]  -0.551   0.067  -0.684  -0.591  -0.551  -0.512  -0.430 1.079
## theta[3]   0.322   0.040   0.247   0.298   0.321   0.345   0.403 1.057
## theta[4]   0.327   0.040   0.254   0.303   0.326   0.351   0.404 1.056
## theta[5]   0.319   0.040   0.244   0.297   0.319   0.342   0.397 1.065
## theta[6]   0.631   0.075   0.494   0.586   0.629   0.674   0.779 1.059
## deviance 178.454  14.156 172.543 174.877 176.880 179.759 188.687 1.002
##          n.eff
## theta[1]    89
## theta[2]    37
## theta[3]    57
## theta[4]    49
## theta[5]    44
## theta[6]    47
## deviance  1800
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 100.2 and DIC = 278.7
## DIC is an estimate of expected predictive error (lower deviance is better).
```

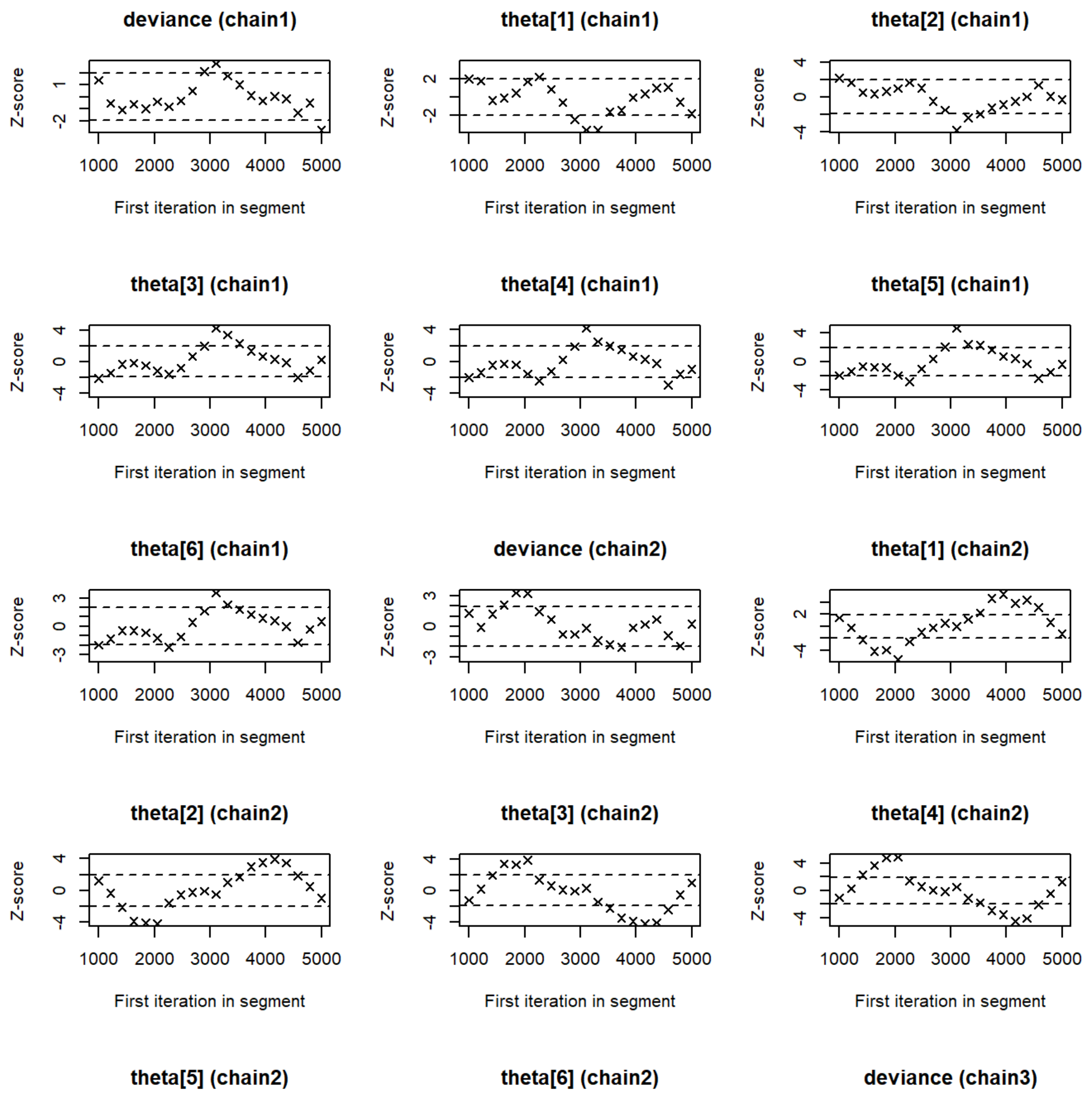
```
modProbitSim.fit$BUGSoutput$summary
```

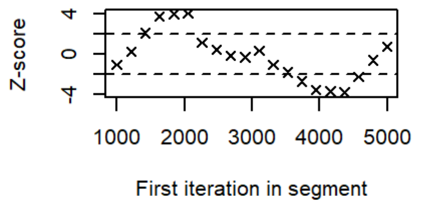
```
##                 mean          sd        2.5%         25%         50%
## deviance 178.4544271 14.15592826 172.5425276 174.8771212 176.8799982
## theta[1]  -0.7373539  0.14144453  -1.0259282  -0.8274324  -0.7351210
## theta[2]  -0.5509691  0.06734137  -0.6838848  -0.5909119  -0.5514663
## theta[3]   0.3215523  0.04021609   0.2470976   0.2984132   0.3208913
## theta[4]   0.3268974  0.03986565   0.2539653   0.3030755   0.3261917
## theta[5]   0.3193184  0.03983029   0.2444613   0.2965634   0.3191179
## theta[6]   0.6308121  0.07470180   0.4944162   0.5861214   0.6291318
##                  75%       97.5%     Rhat n.eff
## deviance 179.7592190 188.6867598 1.002372  1800
## theta[1]  -0.6468672  -0.4697787 1.029715    89
## theta[2]  -0.5118459  -0.4299104 1.079382    37
## theta[3]   0.3448132   0.4027540 1.057034    57
## theta[4]   0.3506476   0.4041169 1.056378    49
## theta[5]   0.3420907   0.3974266 1.064690    44
## theta[6]   0.6738723   0.7794224 1.058762    47
```

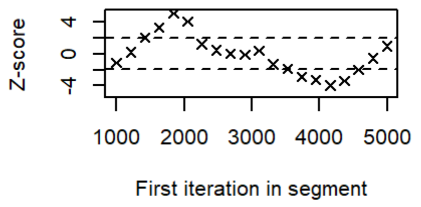
```
chainArray <- modProbitSim.fit$BUGSoutput$sims.array

# Plots with BayesPlot
#bayesplot::mcmc_combo(chainArray)
bayesplot::mcmc_acf(chainArray)
```

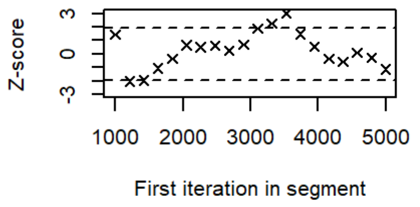


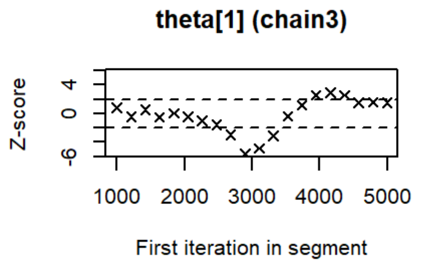
```
# Diagnostic with coda
coda.fit <- as.mcmc(modProbitSim.fit)

coda::geweke.plot(coda.fit)
```

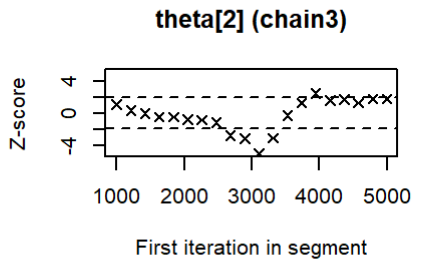

deviance (chain1)
theta[1] (chain1)
theta[2] (chain1)
theta[3] (chain1)
theta[4] (chain1)
theta[5] (chain1)
theta[6] (chain1)
deviance (chain2)
theta[1] (chain2)
theta[2] (chain2)
theta[3] (chain2)
theta[4] (chain2)
theta[5] (chain2)
theta[6] (chain2)
deviance (chain3)
Z-score
First iteration in segment
1000
2000
3000
4000
5000

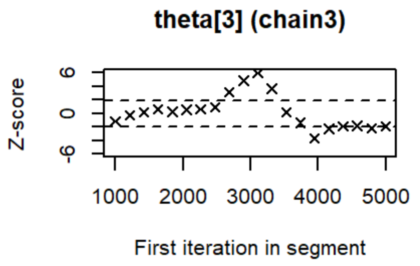

Z-score
4
0
-4
1000
2000
3000
4000
5000
First iteration in segment

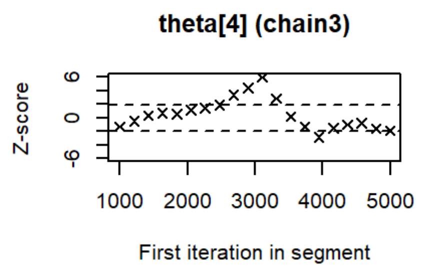

Z-score
4
0
-4
1000
2000
3000
4000
5000
First iteration in segment

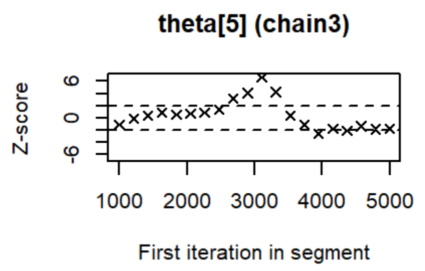

Z-score
3
0
-3
1000
2000
3000
4000
5000
First iteration in segment

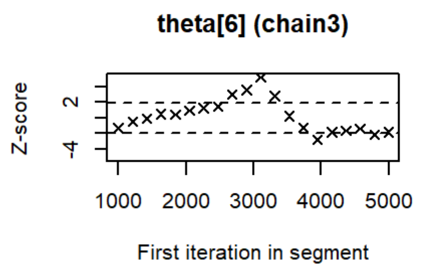

theta[1] (chain3)
Z-score
4
0
-6
1000
2000
3000
4000
5000
First iteration in segment

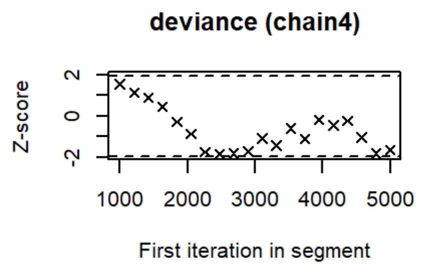

theta[2] (chain3)
Z-score
4
0
-4
1000
2000
3000
4000
5000
First iteration in segment

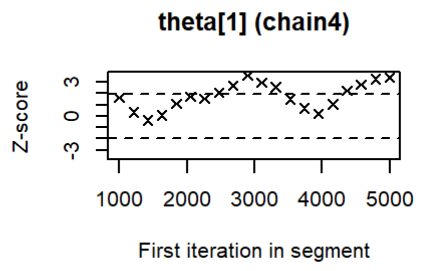

theta[3] (chain3)
Z-score
6
0
-6
1000
2000
3000
4000
5000
First iteration in segment

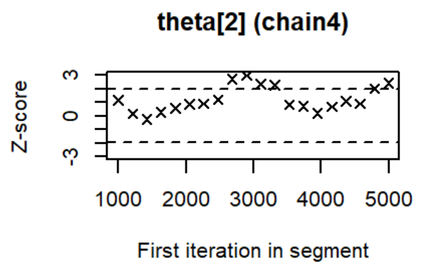

theta[4] (chain3)
Z-score
6
0
-6
1000 2000 3000 4000 5000
First iteration in segment

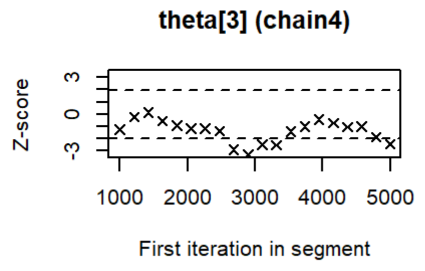

theta[5] (chain3)
Z-score
6
0
-6
1000 2000 3000 4000 5000
First iteration in segment

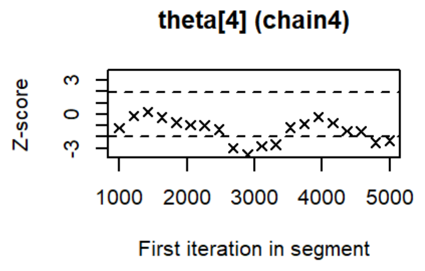

theta[6] (chain3)
Z-score
2
-4
1000 2000 3000 4000 5000
First iteration in segment

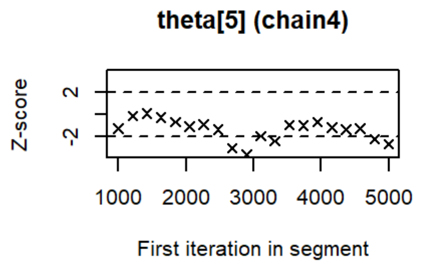

deviance (chain4)
Z-score
2
0
-2
1000 2000 3000 4000 5000
First iteration in segment

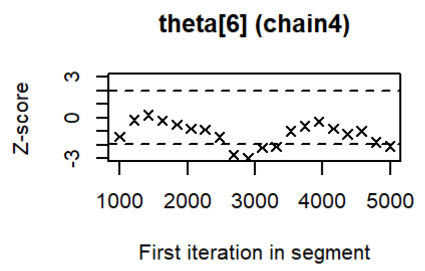

theta[1] (chain4)
Z-score
3
0
-3
1000 2000 3000 4000 5000
First iteration in segment


theta[2] (chain4)
Z-score
3
0
-3
1000 2000 3000 4000 5000
First iteration in segment

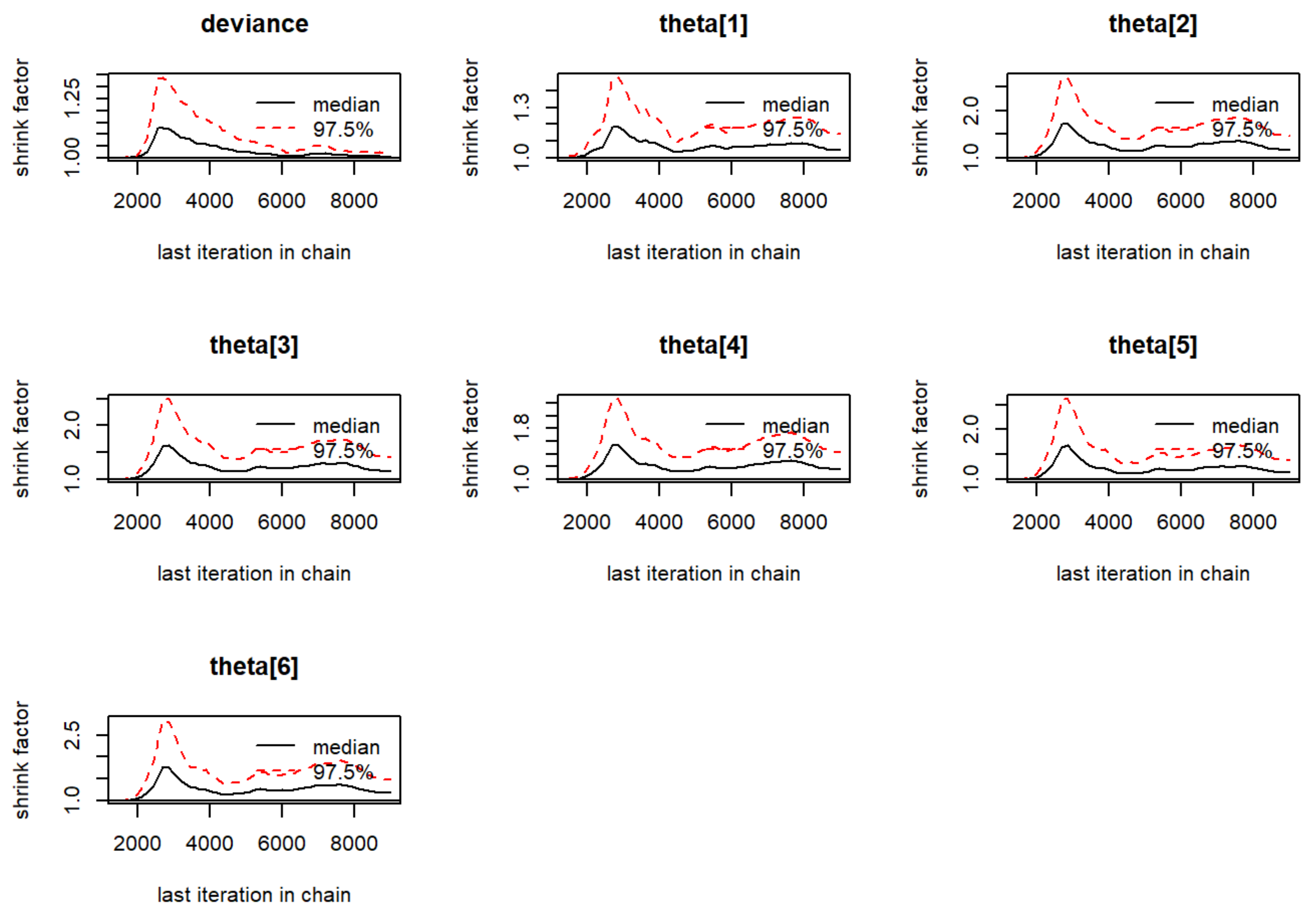

theta[3] (chain4)
Z-score
3
0
-3
1000 2000 3000 4000 5000
First iteration in segment


theta[4] (chain4)
Z-score
3
0
-3
1000 2000 3000 4000 5000
First iteration in segment


theta[5] (chain4)
Z-score
2
-2
1000 2000 3000 4000 5000
First iteration in segment



```
coda::gelman.plot(coda.fit)
```



```
# Manipulating the chain -------------------------------------------------------------

chainMat <- modProbitSim.fit$BUGSoutput$sims.matrix


# Point estimates
(theta.hat_sim.jags <- colMeans(chainMat))
```

```
##    deviance    theta[1]    theta[2]    theta[3]    theta[4]    theta[5]
## 178.4544271  -0.7373539  -0.5509691   0.3215523   0.3268974   0.3193184
##    theta[6]
##   0.6308121
```

```
# Intervals
cred <- 0.95
(theta.ET_sim.jags <- apply(chainMat, 2, quantile, prob=c((1-cred)/2, 1-(1-cred)/2)))
```

```
##       deviance   theta[1]   theta[2]  theta[3]  theta[4]  theta[5]
## 2.5%  172.5425 -1.0259282 -0.6838848 0.2470977 0.2539653 0.2444613
## 97.5% 188.6868 -0.4697787 -0.4299104 0.4027540 0.4041169 0.3974266
##        theta[6]
## 2.5%  0.4944162
## 97.5% 0.7794224
```

```
# What about the HPD?
(theta.HPD_sim.jags <- coda::HPDinterval(as.mcmc(chainMat)))
```

```
##                lower       upper
## deviance 171.6660105 185.7198036
## theta[1]  -1.0288525  -0.4731147
## theta[2]  -0.6989373  -0.4481607
## theta[3]   0.2541994   0.4082155
## theta[4]   0.2553211   0.4047660
## theta[5]   0.2550307   0.4071078
## theta[6]   0.5089667   0.7826548
## attr(,"Probability")
## [1] 0.95
```

```
cat('parameters via MCMC with original data are', theta, 'parameters via MCMC with simulated data are:',theta.hat
_sim.jags[-1])
```

```
## parameters via MCMC with original data are -0.5217235 -0.452429 0.2557826 0.2678606 0.2603645 0.4959219 parame
ters via MCMC with simulated data are: -0.7373539 -0.5509691 0.3215523 0.3268974 0.3193184 0.6308121
```

```
#Error approximation
errors=sapply(modProbitSim.fit$BUGSoutput$sd, function(x) x^2/(modProbitSim.fit$n.iter-1000))
cat('approximation errors are ',errors$theta)
```

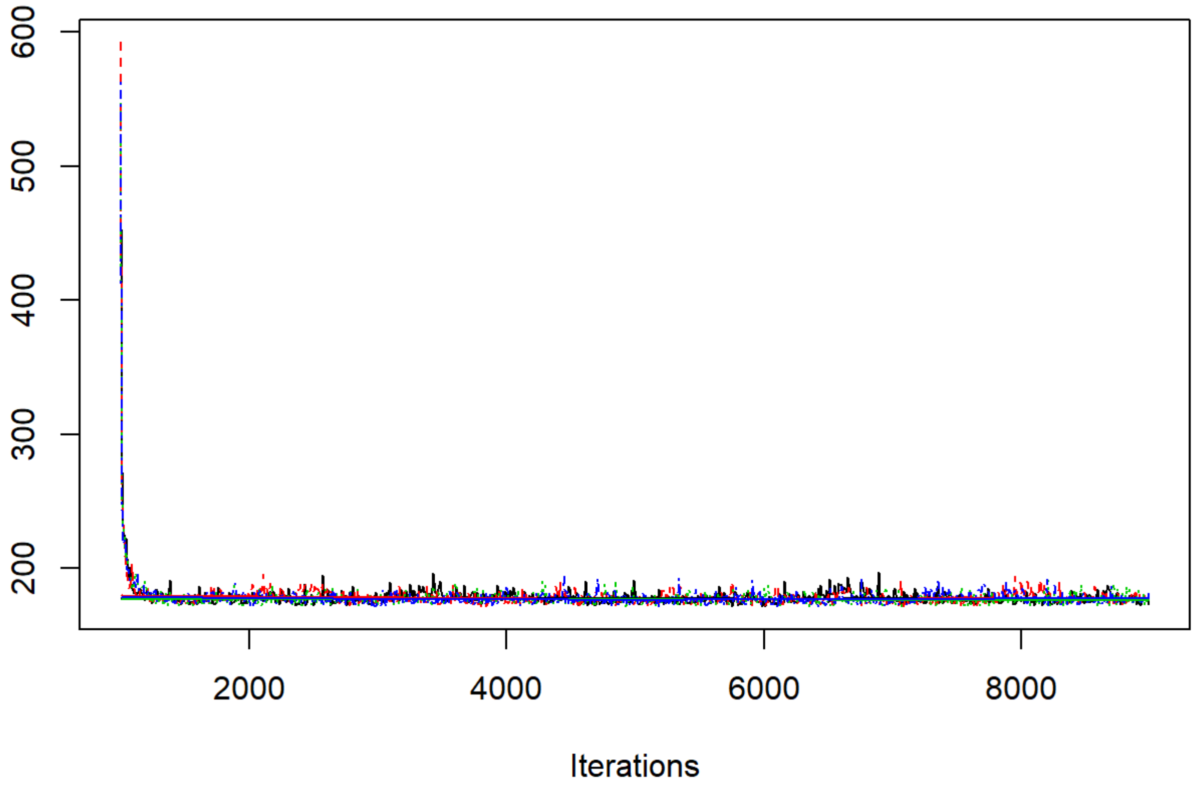
```
## approximation errors are  2.500819e-06 5.668575e-07 2.021667e-07 1.986588e-07 1.983065e-07 6.975449e-07
```

```
# Parameter uncertainty
cat('uncertainty for parameters',abs(modProbitSim.fit$BUGSoutput$sd$theta/theta.hat_sim.jags[-1]))
```

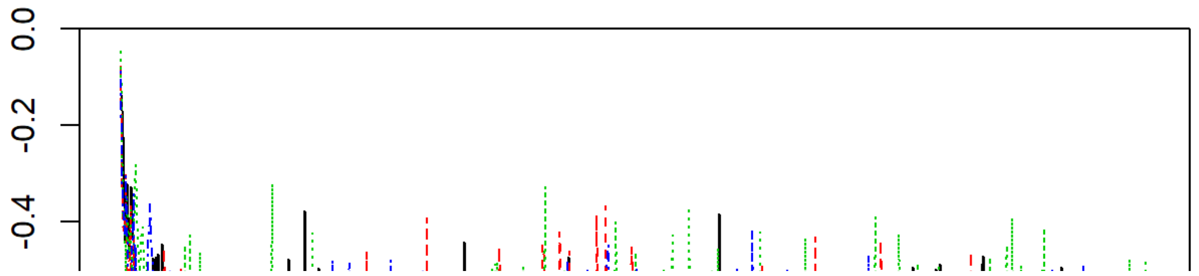
```
## uncertainty for parameters 0.1918272 0.1222235 0.1250686 0.1219516 0.1247353 0.1184216
```

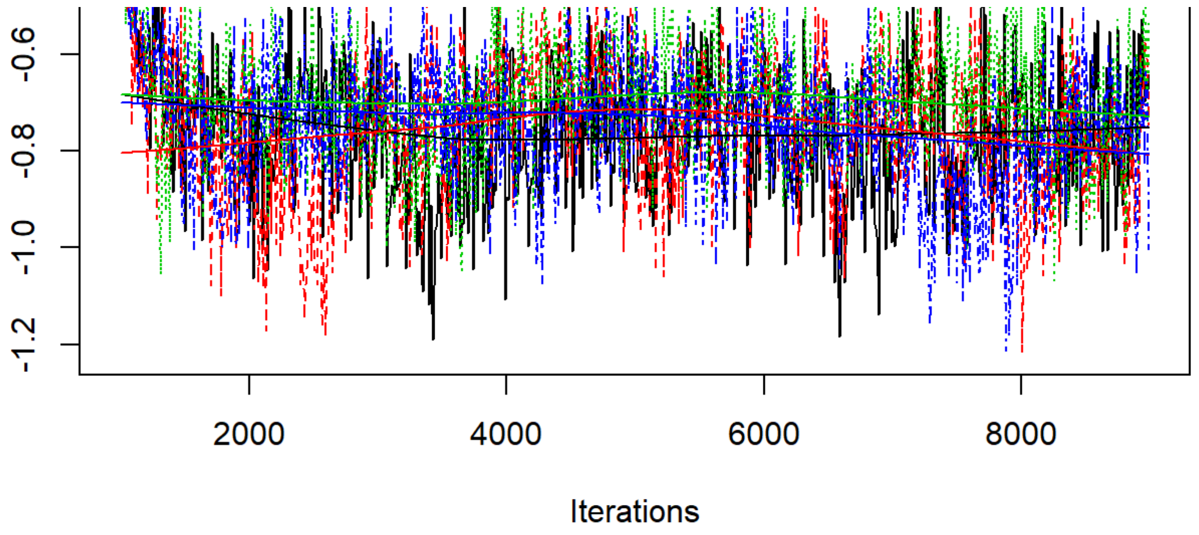
```
#Traceplot
traceplot(as.mcmc(modProbitSim.fit))
```

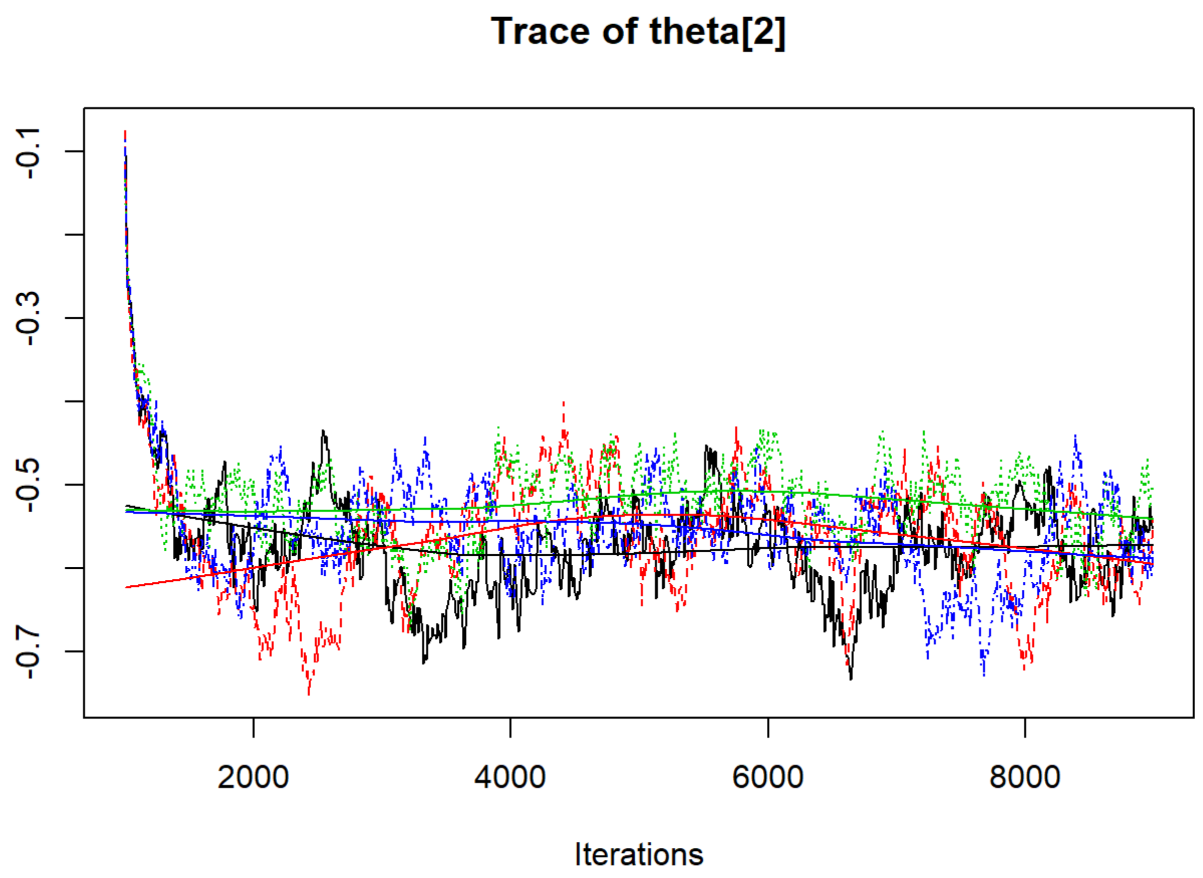
Trace of deviance

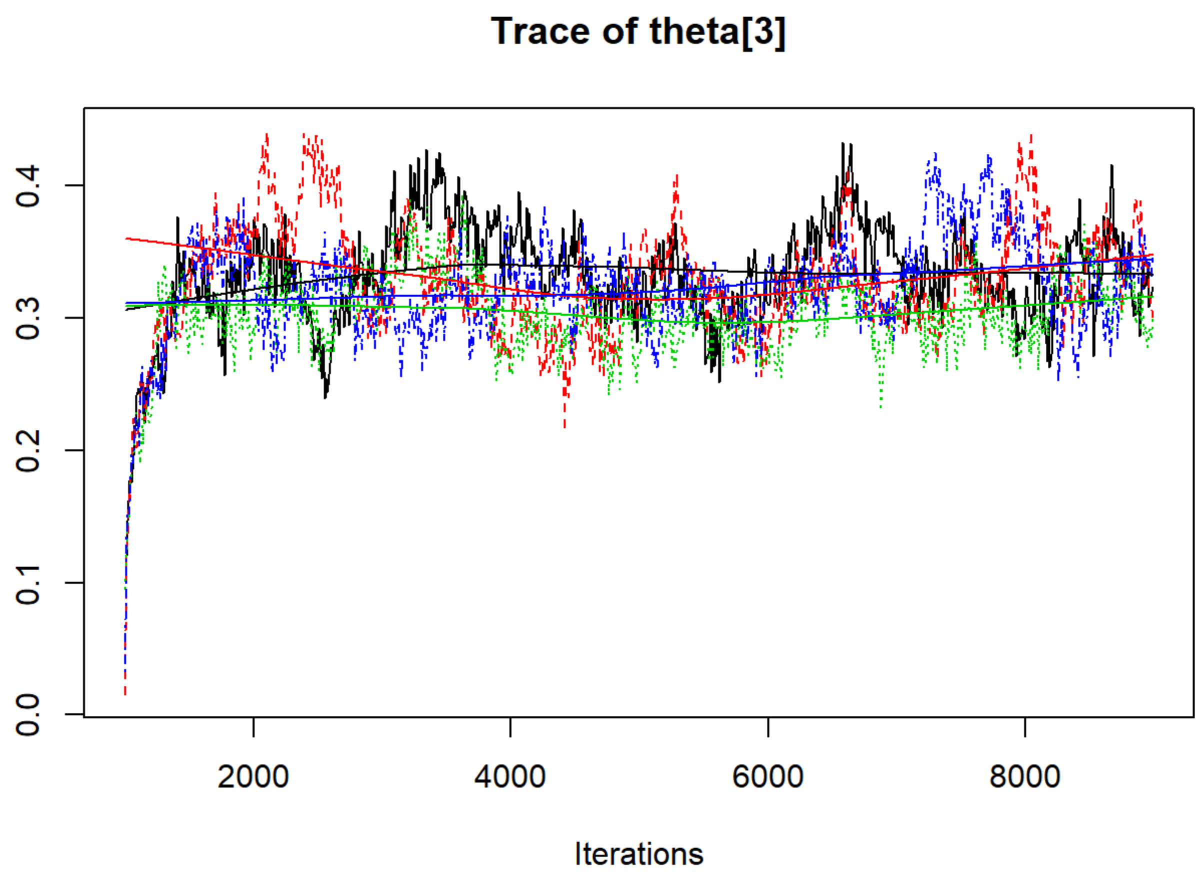


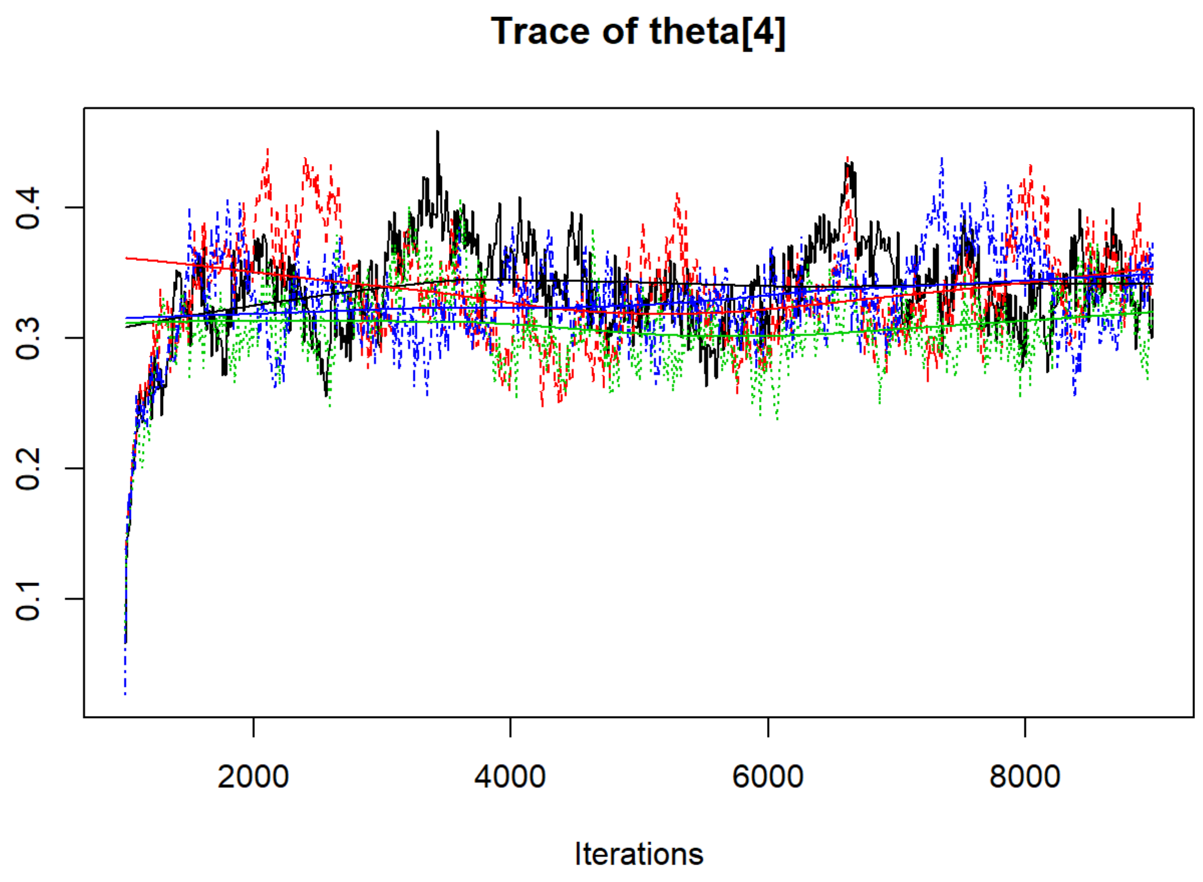
Trace of theta[1]



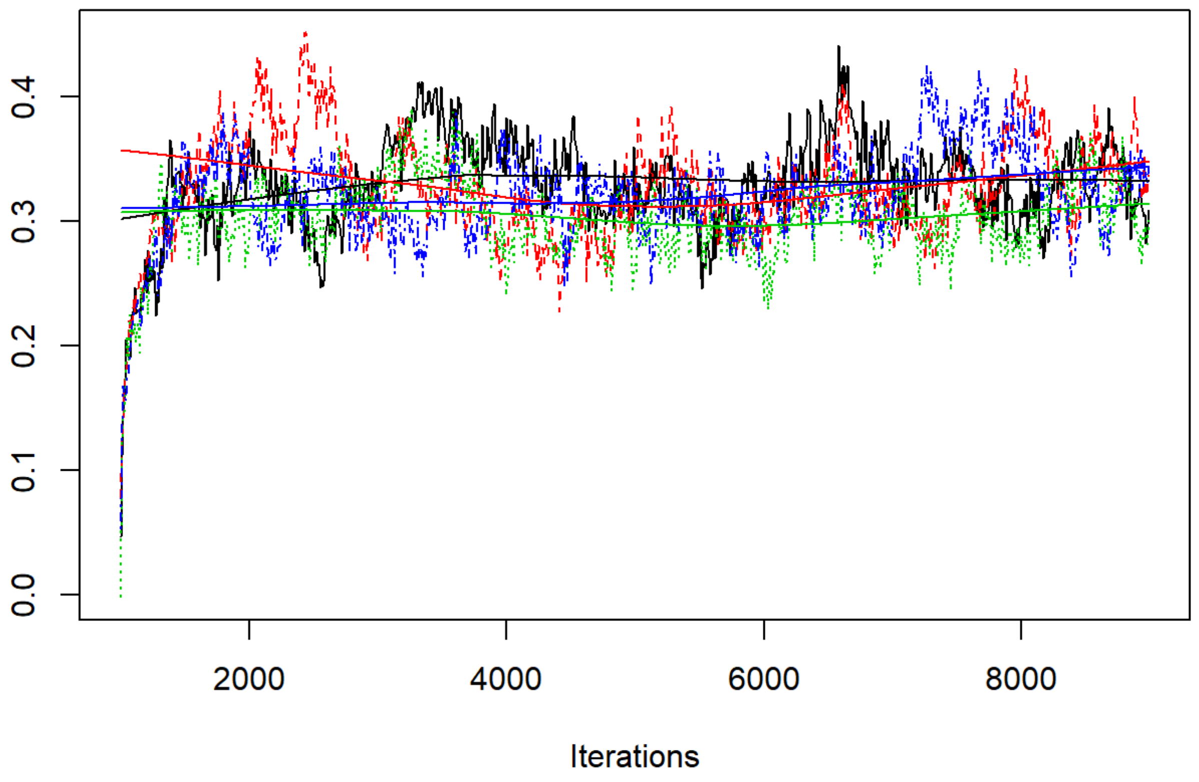

-0.6
-0.8
-1.0
-1.2
2000
4000
6000
8000
Iterations

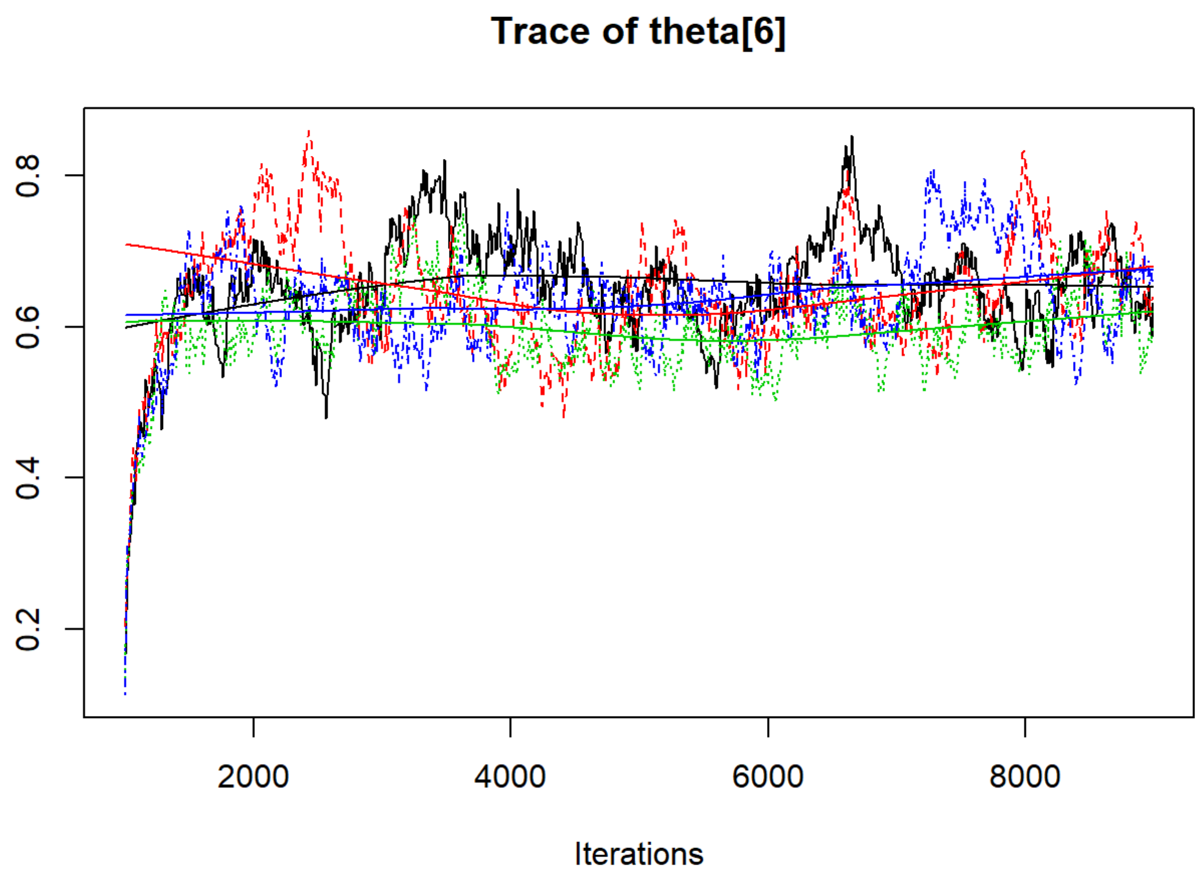

Trace of theta[2]
-0.1
-0.3
-0.5
-0.7
2000
4000
6000
8000
Iterations


Trace of theta[3]
0.0
0.1
0.2
0.3
0.4
2000
4000
6000
8000
Iterations

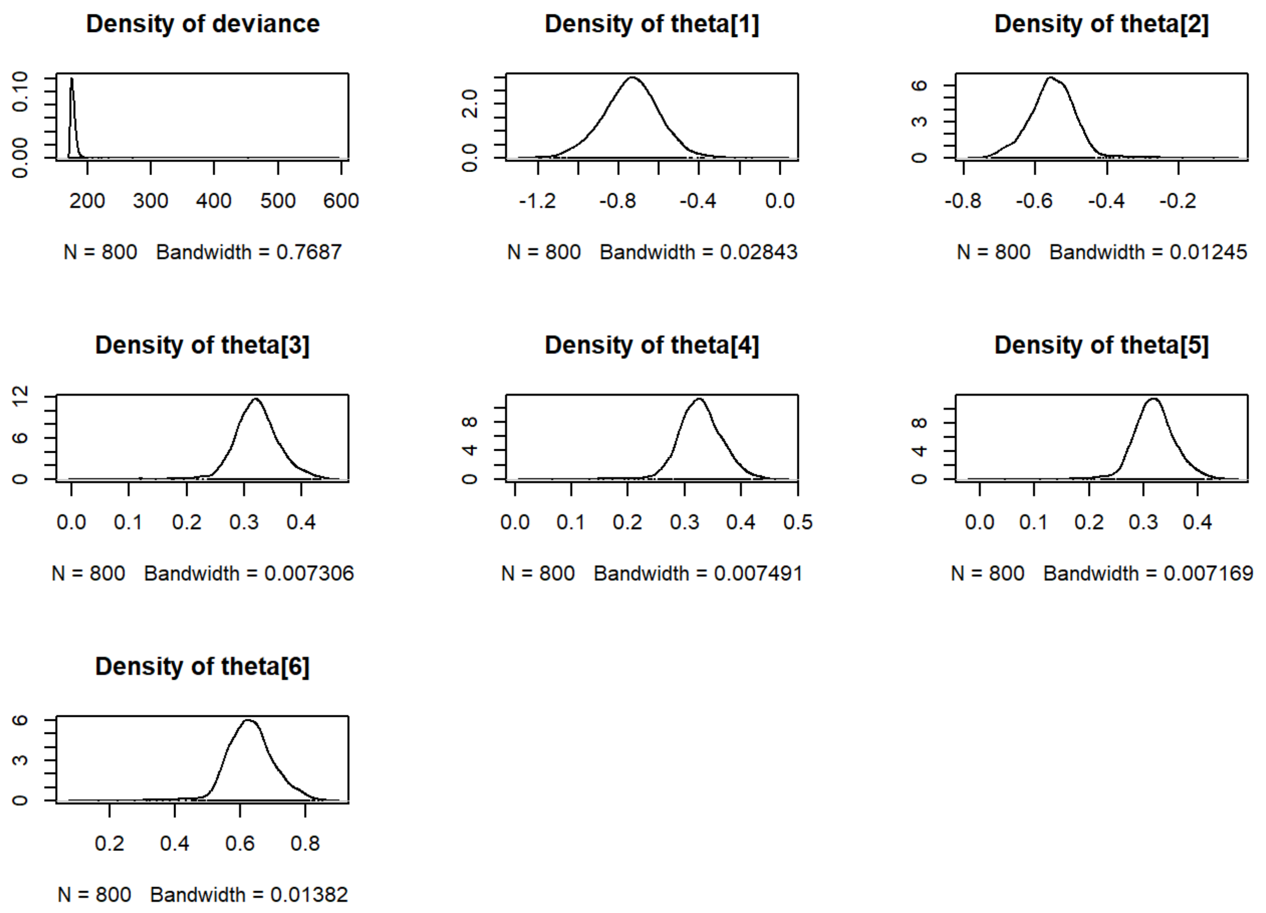

Trace of theta[4]
0.1
0.2
0.3
0.4
2000
4000
6000
8000
Iterations

Trace of theta[5]





```
# Density plots
plot(as.mcmc(modProbitSim.fit),density = T,trace = F)
```



```
#Gelmna-Rubin Test

gelman.diag(as.mcmc(modProbitSim.fit))
```

```
## Potential scale reduction factors:
##
##          Point est. Upper C.I.
## deviance       1.01       1.02
## theta[1]       1.05       1.14
## theta[2]       1.17       1.47
## theta[3]       1.15       1.42
## theta[4]       1.16       1.44
## theta[5]       1.14       1.39
## theta[6]       1.18       1.49
##
## Multivariate psrf
##
## 1.16
```

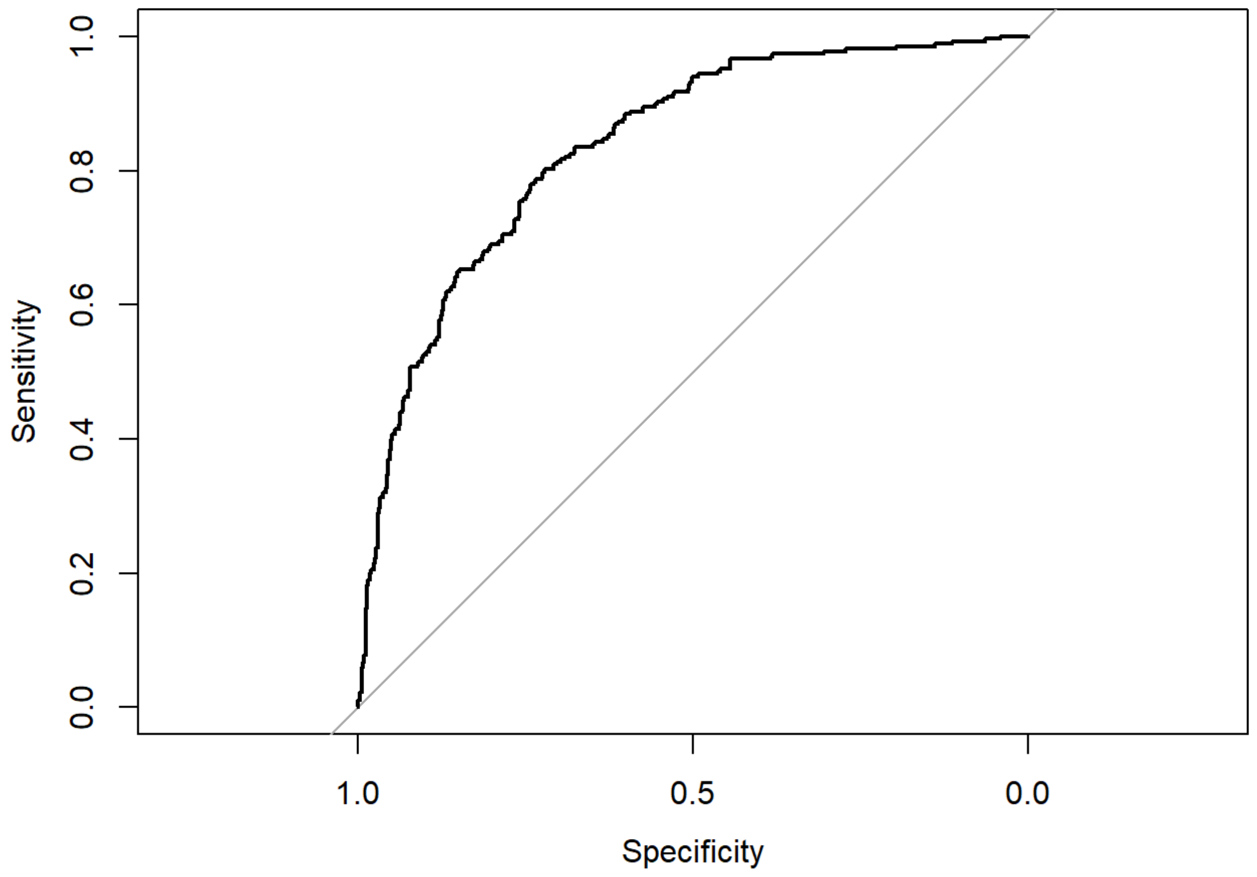
```
#gelman.plot(as.mcmc(modLogit.fit))

#Heidelberger-Welch Test
heidel.diag(as.mcmc(modProbitSim.fit))
```

```
## [[1]]
##
##          Stationarity start     p-value
##          test         iteration
## deviance passed       81        0.7851
## theta[1] passed       81        0.2738
## theta[2] passed        1        0.1397
## theta[3] passed        1        0.1675
## theta[4] passed        1        0.0806
## theta[5] passed        1        0.1074
## theta[6] passed        1        0.1636
##
##          Halfwidth Mean    Halfwidth
##          test
## deviance passed    177.978 0.7225
## theta[1] passed     -0.773 0.0199
## theta[2] passed     -0.571 0.0258
## theta[3] passed      0.332 0.0162
## theta[4] passed      0.338 0.0142
## theta[5] passed      0.329 0.0164
## theta[6] passed      0.652 0.0294
```

```
##
## [[2]]
##
##          Stationarity start     p-value
##          test         iteration
## deviance passed       81         0.111
## theta[1] passed        1         0.101
## theta[2] passed        1         0.371
## theta[3] passed        1         0.304
## theta[4] passed        1         0.302
## theta[5] passed        1         0.280
## theta[6] passed        1         0.329
##
##          Halfwidth Mean    Halfwidth
##          test
## deviance passed    177.965 0.9157
## theta[1] passed     -0.753 0.0355
## theta[2] passed     -0.561 0.0304
## theta[3] passed      0.328 0.0186
## theta[4] passed      0.333 0.0173
## theta[5] passed      0.326 0.0181
## theta[6] passed      0.643 0.0358
##
## [[3]]
##
##          Stationarity start     p-value
##          test         iteration
## deviance passed       81         0.2014
## theta[1] passed        1         0.1195
## theta[2] passed        1         0.2280
## theta[3] passed        1         0.0970
## theta[4] passed        1         0.0833
## theta[5] passed        1         0.1795
## theta[6] passed        1         0.2656
##
##          Halfwidth Mean    Halfwidth
##          test
## deviance passed    177.053 0.25353
## theta[1] passed     -0.693 0.02722
## theta[2] passed     -0.519 0.01896
## theta[3] passed      0.303 0.00961
## theta[4] passed      0.308 0.00949
## theta[5] passed      0.302 0.01004
## theta[6] passed      0.595 0.02098
##
## [[4]]
##
##          Stationarity start     p-value
##          test         iteration
## deviance passed       81        0.3905
## theta[1] passed       81        0.1284
## theta[2] passed        1        0.1189
## theta[3] passed        1        0.0568
## theta[4] passed       81        0.1940
## theta[5] passed        1        0.1124
## theta[6] passed        1        0.1044
##
##          Halfwidth Mean    Halfwidth
##          test
## deviance passed    177.223 0.5225
## theta[1] passed     -0.753 0.0330
## theta[2] passed     -0.552 0.0251
## theta[3] passed      0.323 0.0136
## theta[4] passed      0.332 0.0115
## theta[5] passed      0.320 0.0140
## theta[6] passed      0.633 0.0286
```

```
#ROC Curve of point estimates

X1=X[,1]
X2=X[,2]
X3=X[,3]
X4=X[,4]
X5=X[,5]
```

```
#re-plug the original data


mcmc_fit_values_sim=theta.hat_sim.jags[-1][1]+theta.hat_sim.jags[-1][2]*X[,1]+theta.hat_sim.jags[-1][3]*X[,2]+theta.hat_sim.jags[-1][4]*X[,3]+theta.hat_sim.jags[-1][5]*X[,4]+theta.hat_sim.jags[-1][6]*X[,5]


roc(Y,pnorm(mcmc_fit_values_sim),plot = T)
```



```
##
## Call:
## roc.default(response = Y, predictor = pnorm(mcmc_fit_values_sim),     plot = T)
##
## Data: pnorm(mcmc_fit_values_sim) in 500 controls (Y 0) < 268 cases (Y 1).
## Area under the curve: 0.8355
```

Although perhaps not optimal, our chains seem to converge,based on the plots, the R-hat values of the parameters and the tests we performed,similar as before.The confidence intervals for our parameters are small(the biggest interval is for the theta1 term).Our point estimates produce an ROC curve with similar area under the curve as for the one we got for the original data.We can conclude that both of our models are consistent, since we recover the same parameters for original and simulated data,with their point estimates and intervals similar to the ones we get by the Frequentist Inference. The logit model has a smaller pD penalty (22.4 vs 30.5 for the probit model),although both models have the same number of parameters.Furthermore, the DIC of the logit model is lower,not by a large margin though(760.3 vs 770.6).The Deviance Information Criterior (DIC) can be computed as:

$$DIC = p_D + \bar{D} = \frac{1}{2}\widehat{var}(D(\theta)) - 2log(L(y|\theta)) + C$$

the smaller the DIC is for one model, the better…

We can claim that the 1st model is marginally better than the second one,although we expect both of them to have predictive ability and perform similarly on unseen data.

References

- Heidelberger, P. and Welch, P.D. (1981). “A Spectral Method for Confidence Interval Generation and Run Length Control in Simulations”. Comm. ACM., 24, p. 233–245.
- Gelman, A and Rubin, DB (1992) Inference from iterative simulation using multiple sequences, Statistical Science, 7, 457-511.
- Ioannis Ntzoufras,(2011) “Bayesian Modeling Using WinBUGS”,10-11
- Peter D. Hoff, “A First Course in Bayesian Statistical Methods”