732A91 Lab 4

Fanny Karelius (fanka300), Milda Poceviciute (milpo192)

15 May 2018

Question 1: Poisson regression - the MCMC way

The y_ith $(i = 1, ..., n)$ count of for the ith observation in the sample follows the following regression model:

$$y_i|\beta \sim Poisson[exp(x_i^T\beta)]$$

where x_i is a p-dimensional vector of covariates.

The data set contains observations form 1000 eBay auctions of coins. The response variable is nBids (the number of bids in each auction). The covariates are Const (intercept), PowerSeller, VerifyID, Sealed, MinBlem, MajBlem, LargNeg, LogBook and MinBidShare.

a)

```
data <- read.table("eBayNumberOfBidderdata.dat", header = TRUE)
c <- c(1,3:ncol(data))
data_noC <- data[,c]
glm_fit <- glm(nBids~., family=poisson, data_noC)
summary(glm_fit)
```

```
##
## Call:
## glm(formula = nBids ~ ., family = poisson, data = data_noC)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.5800  -0.7222  -0.0441   0.5269   2.4605
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.07244    0.03077  34.848  < 2e-16 ***
## PowerSeller -0.02054    0.03678  -0.558   0.5765
## VerifyID    -0.39452    0.09243  -4.268 1.97e-05 ***
## Sealed       0.44384    0.05056   8.778  < 2e-16 ***
## Minblem     -0.05220    0.06020  -0.867   0.3859
## MajBlem     -0.22087    0.09144  -2.416   0.0157 *
## LargNeg      0.07067    0.05633   1.255   0.2096
## LogBook     -0.12068    0.02896  -4.166 3.09e-05 ***
## MinBidShare -1.89410    0.07124 -26.588  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 2151.28  on 999  degrees of freedom
## Residual deviance:  867.47  on 991  degrees of freedom
## AIC: 3610.3
##
## Number of Fisher Scoring iterations: 5
```

From the summary we see that the Intercept, VerifyID, Sealed, MajBlem, LogBook and MinBidShare are significant covariates.

b)

Here we do Bayesian analysis of the Poisson regression. The prior is $\beta \sim N(0, 100 \cdot (X^TX)^{-1})$, where X is a $n \times p$ covariate matrix, which is called Zellner's g-prior. We assume that the posterior density is approximately multivariate normal:

$$\beta | y \sim N(\tilde{\beta}, J_y^{-1}(\tilde{\beta}))$$

where $\tilde{\beta}$ is the posterior mode and $J_y(\tilde{\beta})$ is the negative Hessian at the posterior mode.

```r
library(mvtnorm)

log_poisson <- function(beta, y, x){
  x <- as.matrix(x)
  xtx <- solve(t(x)%*%x)
  y <- as.matrix(y)
  lin_pred <- x%*%beta
  loglik <- sum(y*lin_pred-exp(lin_pred))/sum(log(factorial(y)))
  logprior <- dmvnorm(beta, mean=as.vector(rep(0,ncol(x))), sigma=100*xtx, log = TRUE)
  logpost <- loglik+logprior
  logpost
}

betas_init <- as.vector(rep(0,ncol(data)-1))
y <- data$nBids
x<- data[,2:ncol(data)]
optim_fit <- optim(betas_init, log_poisson, gr=NULL, y, x,
                   method=c("BFGS"),control=list(fnscale=-1), hessian=TRUE)

beta_tilde <- optim_fit$par
hessian <- -1*optim_fit$hessian
inv_hessian <- solve(hessian)

post_approx <- rmvnorm(n=1000, mean=beta_tilde, sigma=inv_hessian)
colnames(post_approx) <- colnames(x)
phi_b <- exp(post_approx)
```

```
## [1] "The posterior mode:"

## [1]  0.072645168 -0.002363413 -0.016290961  0.041358118 -0.009751628
## [6] -0.029472899  0.008762620 -0.002913510 -0.142679215

## [1] "The negative inverse Hessian:"

##               [,1]        [,2]        [,3]        [,4]        [,5]
##  [1,]  0.277961151 -0.24888154 -0.07709695 -0.09948810 -0.15502241
##  [2,] -0.248881540  0.47523863 -0.09261390 -0.06771881  0.04657194
##  [3,] -0.077096951 -0.09261390  1.90532349 -0.07224421  0.01640144
##  [4,] -0.099488096 -0.06771881 -0.07224421  1.15668610  0.12051833
##  [5,] -0.155022408  0.04657194  0.01640144  0.12051833  1.17985533
##  [6,] -0.131675787 -0.03682095  0.10972495  0.15358135  0.13032112
##  [7,] -0.171931301  0.05579491  0.13735299  0.12713033  0.03627697
##  [8,] -0.008658968  0.05067481 -0.20219725 -0.01758610  0.03546605
##  [9,]  0.097638506 -0.17871020 -0.29922724  0.07954534  0.03369626
##              [,6]         [,7]         [,8]         [,9]
##  [1,] -0.13167579 -0.171931301 -0.008658968  0.097638506
##  [2,] -0.03682095  0.055794911  0.050674812 -0.178710201
##  [3,]  0.10972495  0.137352992 -0.202197252 -0.299227238
##  [4,]  0.15358135  0.127130334 -0.017586105  0.079545339
##  [5,]  0.13032112  0.036276967  0.035466048  0.033696256
##  [6,]  2.88186700  0.161995542 -0.031398337  0.145251562
##  [7,]  0.16199554  1.305247679 -0.075269365  0.002281561
##  [8,] -0.03139834 -0.075269365  0.275074867  0.242765592
##  [9,]  0.14525156  0.002281561  0.242765592  1.034958424
```

c)

The Metropolis algorithm simulates from the actual posterior of β. We use

$$\theta_p|\theta_c \sim N(\theta_c, \tilde{c} \cdot \Sigma)$$

where $\Sigma = J_y^{-1}(\tilde{\beta})$, to draw θs. The value $\tilde{c}$ is a tuning parameter. The acceptance rate is given by

$$\alpha = \min\{1, \frac{p(\theta_p|y)}{p(\theta_c|y)} = \exp[\log(p(\theta_p|y)) - log(p(\theta_c|y))]\}$$

. We aim to have an acceptance rate of the Metropolis algorithm around 30%.

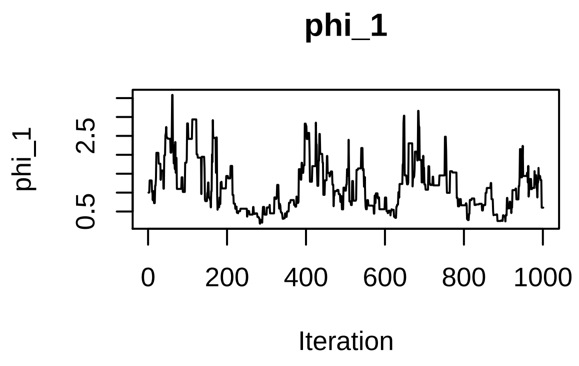
```r
metropolis <- function(n, c, sigma, logpostfun, theta,...){
  thetas <- matrix(nrow=n+1, ncol=length(theta))
  thetas[1,]<-theta
  temp1 <-logpostfun(thetas[1,],...)
  acc_prob <- vector(length=n)
  acc_prob[1] <- 0
  for(i in 1:n){
    temp_theta<-rmvnorm(n=1, thetas[i,], c*sigma)
    temp2 <-logpostfun(as.vector(temp_theta),...)
    acc_prob[i+1] <- min(1,exp(temp2-temp1))
    u <- runif(n = 1,0,1)
    if(u>acc_prob[i+1]){
      thetas[i+1,]<-thetas[i,]
    }
    else{
      thetas[i+1,] <- temp_theta
      temp1 <- temp2
    }
  }
  data.frame(thetas, "acc.prob"=acc_prob)
}
metro2 <- metropolis(1000, c=1, sigma=inv_hessian, log_poisson, theta=betas_init, y=y, x=x)
metro <- metropolis(1000, c=0.5, sigma=inv_hessian, log_poisson, theta=betas_init, y=y, x=x)

acc_prob <- as.vector(metro$acc.prob)
avg_acc <- mean(acc_prob)
```

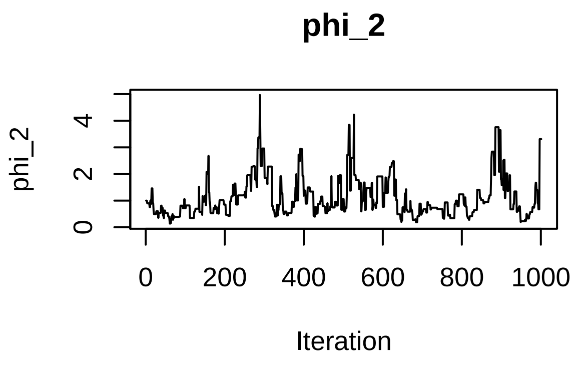
```r
beta_metro <- as.matrix(metro[,1:length(betas_init)])
phi <- exp(beta_metro)
phi_means <- colMeans(phi)
```

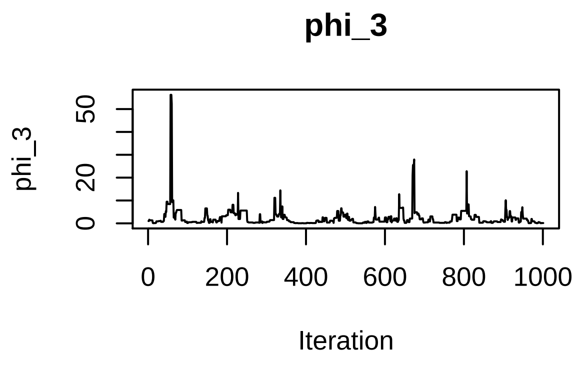
```
## Average acceptance rate with c = 1:  0.1587041
```

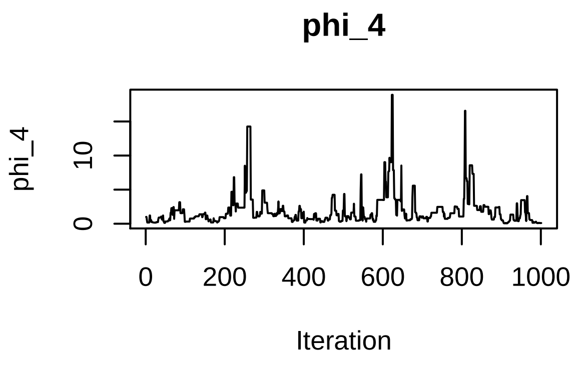
```
##
##  Average acceptance rate with c = 0.5:  0.2981976
```

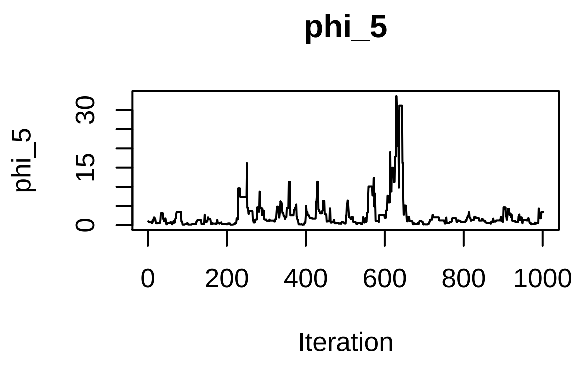
Hence, we conclude that $\tilde{c} = 0.5$ is appropriate tuning parameter in our case.

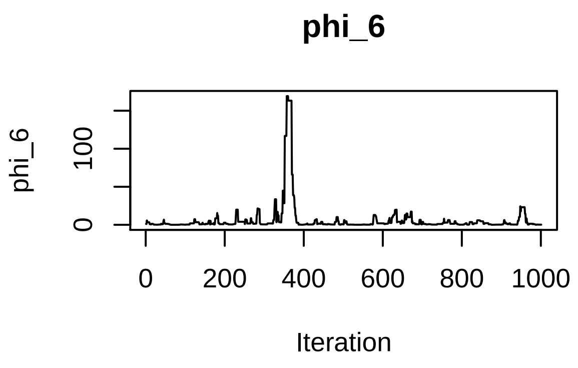

phi_1
phi_1
0.5
2.5
0
200
400
600
800
1000
Iteration

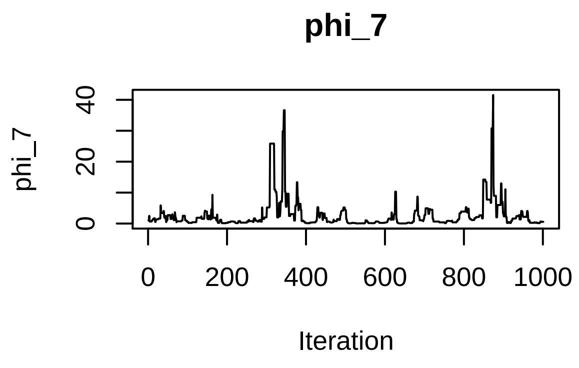

phi_2
phi_2
0
2
4
0
200
400
600
800
1000
Iteration

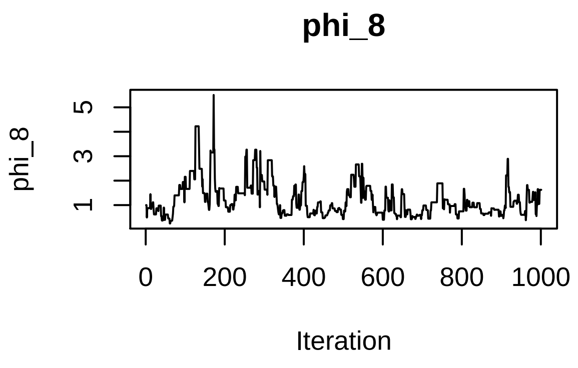

phi_3
phi_3
0
20
50
0
200
400
600
800
1000
Iteration

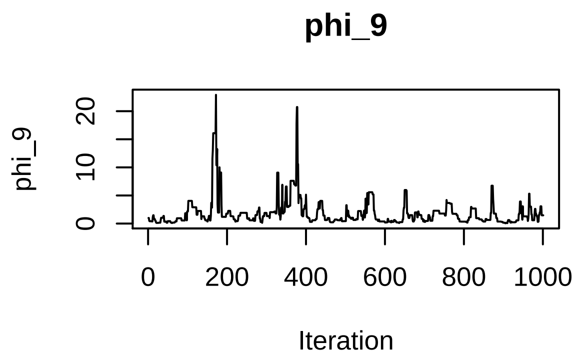

phi_4
phi_4
0
10
0
200
400
600
800
1000
Iteration


phi_5
phi_5
0
15
30
0
200
400
600
800
1000
Iteration

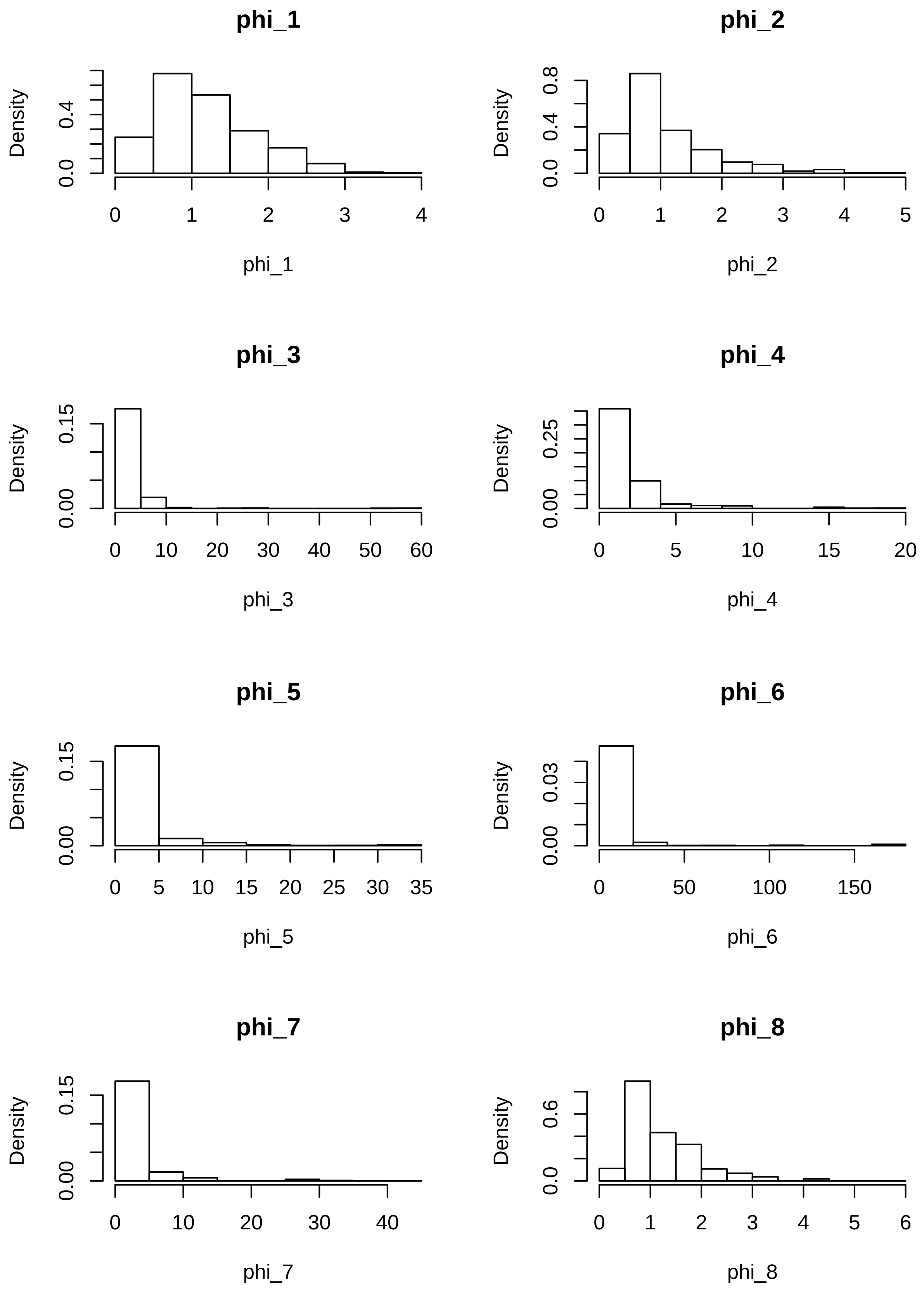

phi_6
phi_6
0
100
0
200
400
600
800
1000
Iteration

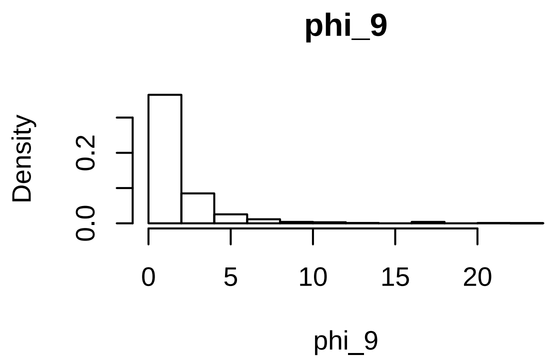

phi_7
phi_7
0
20
40
0
200
400
600
800
1000
Iteration


phi_8
phi_8
1
3
5
0
200
400
600
800
1000
Iteration

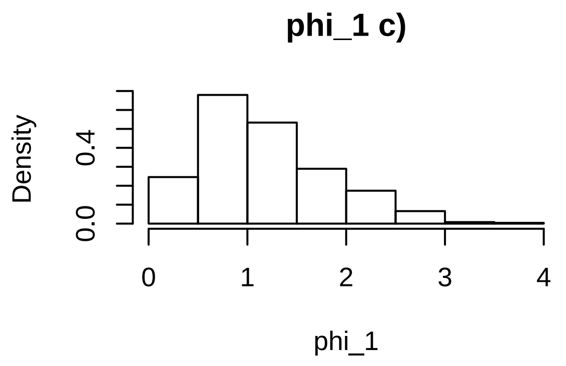


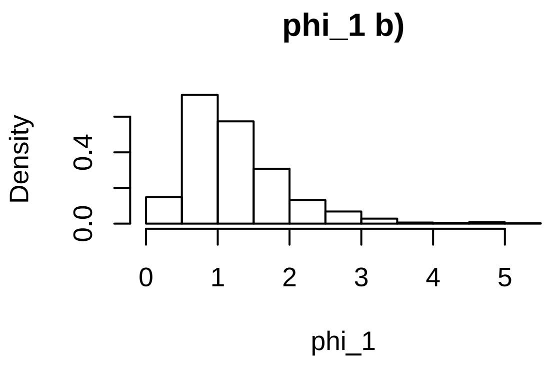
From the plots above of our MCMC process we see that the algorithm is converging ok, because the βs do not get stuck at the same values for longer periods of iterations. The corresponding ϕ_i distributions are represenetd by the histograms:

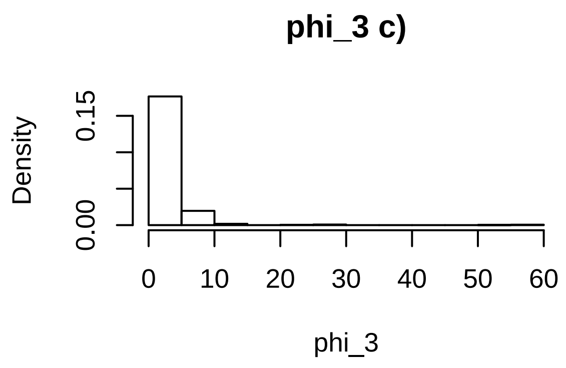

phi_1
Density
0.0
0.4
0
1
2
3
4
phi_1
phi_2
Density
0.0
0.4
0.8
0
1
2
3
4
5
phi_2
phi_3
Density
0.00
0.15
0
10
20
30
40
50
60
phi_3
phi_4
Density
0.00
0.25
0
5
10
15
20
phi_4
phi_5
Density
0.00
0.15
0
5
10
15
20
25
30
35
phi_5
phi_6
Density
0.00
0.03
0
50
100
150
phi_6
phi_7
Density
0.00
0.15
0
10
20
30
40
phi_7
phi_8
Density
0.0
0.6
0
1
2
3
4
5
6
phi_8

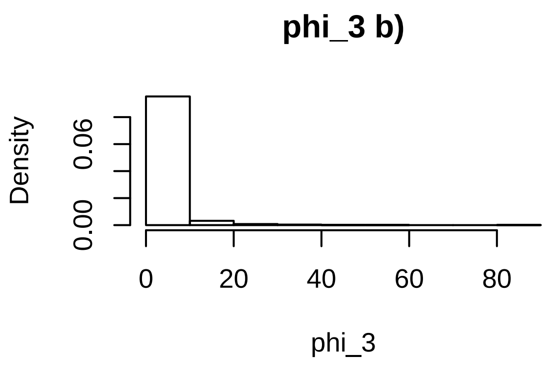


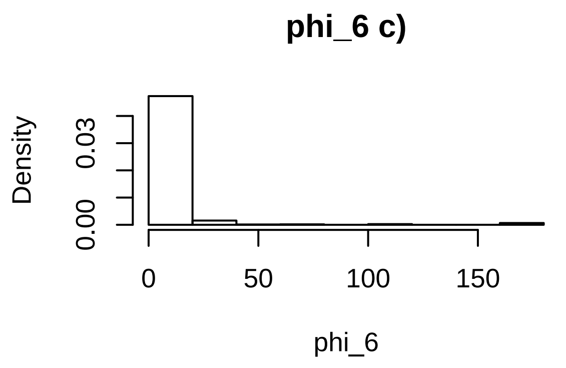
The comparison of simulations from part b) and part c):

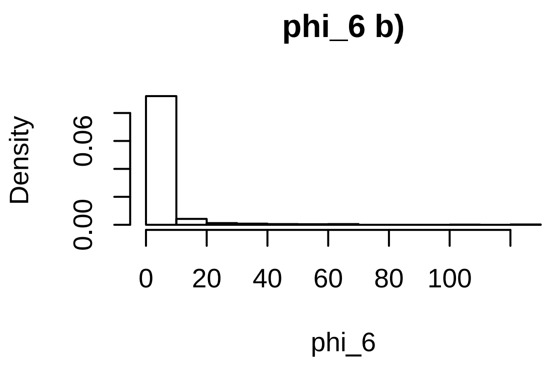

phi_1 c)
Density
0.0
0.4
0
1
2
3
4
phi_1

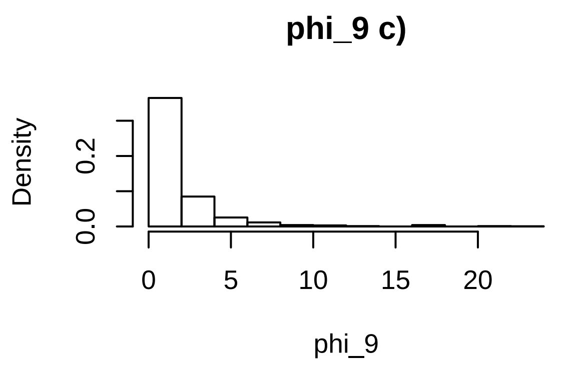

phi_1 b)
Density
0.0
0.4
0
1
2
3
4
5
phi_1

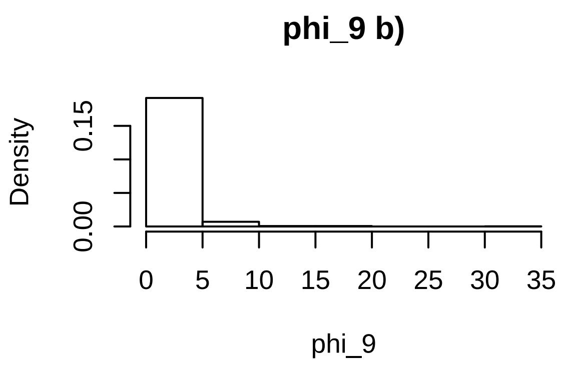

phi_3 c)
Density
0.00
0.15
0
10
20
30
40
50
60
phi_3


phi_3 b)
Density
0.00
0.06
0
20
40
60
80
phi_3


phi_6 c)
Density
0.00
0.03
0
50
100
150
phi_6


phi_6 b)
Density
0.00
0.06
0
20
40
60
80
100
phi_6


phi_9 c)
Density
0.0
0.2
0
5
10
15
20
phi_9

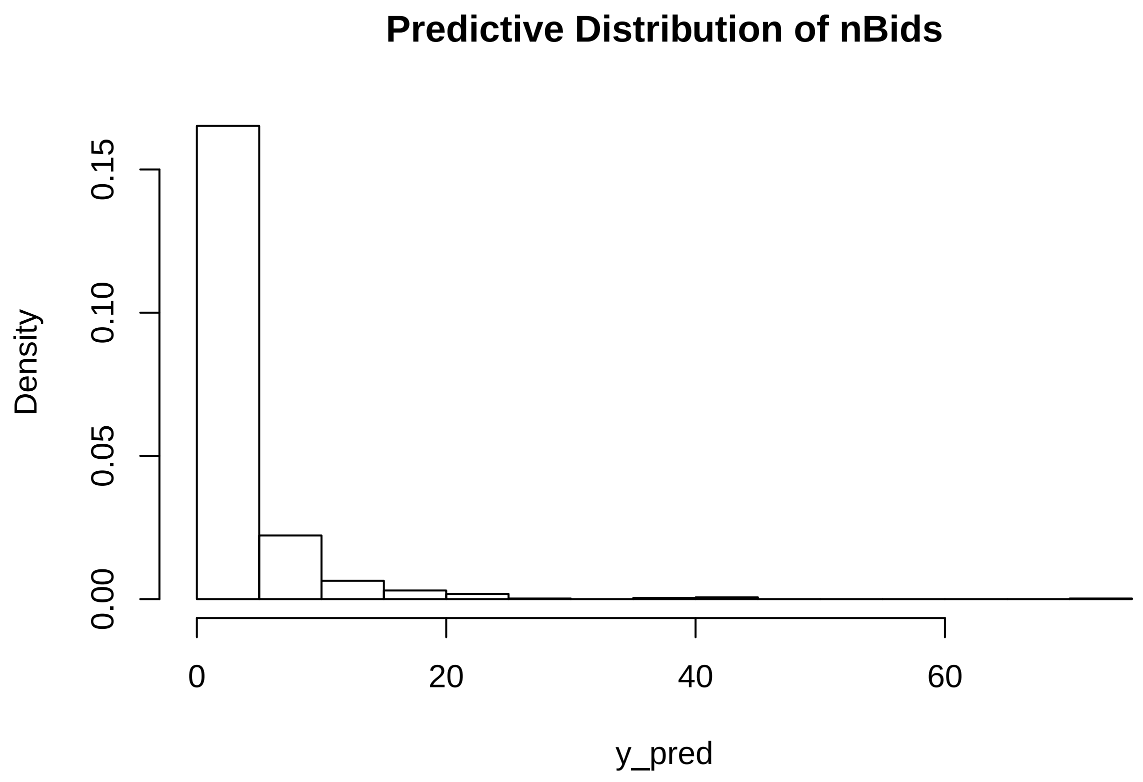

phi_9 b)
Density
0.00
0.15
0
5
10
15
20
25
30
35
phi_9

From visual inspection, we conclude that the majority of ϕ_is are approximated quite well in part b) when compared to draws from the actual distribution.

d)

We simulate from the predictive distribution of the number of bidders (nBids) using the MCMC draws from c). The provided covariates are PowerSeller = 1, VerifyID = 1, Sealed = 1, MinBlem = 0, MajBlem = 0, LargNeg = 0, LogBook = 1, and MinBidShare = 0.5.

```
x_vec <- c(1,1,1,1,0,0,0,1,0.5)
lambda <- exp(x_vec%*%t(beta_metro))
y_mat <- apply(lambda, 1, function(i){rpois(1000, i)})
hist(y_mat, freq = FALSE, xlab="y_pred", main="Predictive Distribution of nBids",breaks = 25)
```



```
prob_y0 <- length(y_mat[y_mat==0])/length(y_mat)
```

```
## Probability of nBids=0:  0.358
```

Appendix

```
data <- read.table("eBayNumberOfBidderdata.dat", header = TRUE)
c <- c(1,3:ncol(data))
data_noC <- data[,c]
glm_fit <- glm(nBids~., family=poisson, data_noC)
```

```r
summary(glm_fit)
library(mvtnorm)

log_poisson <- function(beta, y, x){
  x <- as.matrix(x)
  xtx <- solve(t(x)%*%x)
  y <- as.matrix(y)
  lin_pred <- x%*%beta
  loglik <- sum(y*lin_pred-exp(lin_pred))/sum(log(factorial(y)))
  logprior <- dmvnorm(beta, mean=as.vector(rep(0,ncol(x))), sigma=100*xtx, log = TRUE)
  logpost <- loglik+logprior
  logpost
}

betas_init <- as.vector(rep(0,ncol(data)-1))
y <- data$nBids
x<- data[,2:ncol(data)]
optim_fit <- optim(betas_init, log_poisson, gr=NULL, y, x,
                   method=c("BFGS"),control=list(fnscale=-1), hessian=TRUE)

beta_tilde <- optim_fit$par
hessian <- -1*optim_fit$hessian
inv_hessian <- solve(hessian)

post_approx <- rmvnorm(n=1000, mean=beta_tilde, sigma=inv_hessian)
colnames(post_approx) <- colnames(x)
phi_b <- exp(post_approx)
print("The posterior mode:")
beta_tilde
print("The negative inverse Hessian:")
inv_hessian
metropolis <- function(n, c, sigma, logpostfun, theta,...){
  thetas <- matrix(nrow=n+1, ncol=length(theta))
  thetas[1,]<-theta
  temp1 <-logpostfun(thetas[1,],...)
  acc_prob <- vector(length=n)
  acc_prob[1] <- 0
  for(i in 1:n){
    temp_theta<-rmvnorm(n=1, thetas[i,], c*sigma)
    temp2 <-logpostfun(as.vector(temp_theta),...)
    acc_prob[i+1] <- min(1,exp(temp2-temp1))
    u <- runif(n = 1,0,1)
    if(u>acc_prob[i+1]){
      thetas[i+1,]<-thetas[i,]
    }
    else{
      thetas[i+1,] <- temp_theta
      temp1 <- temp2
    }
  }
  data.frame(thetas, "acc.prob"=acc_prob)
}
metro2 <- metropolis(1000, c=1, sigma=inv_hessian, log_poisson, theta=betas_init, y=y, x=x)
```

```r
metro <- metropolis(1000, c=0.5, sigma=inv_hessian, log_poisson, theta=betas_init, y=y, x=x)

acc_prob <- as.vector(metro$acc.prob)
avg_acc <- mean(acc_prob)

beta_metro <- as.matrix(metro[,1:length(betas_init)])
phi <- exp(beta_metro)
phi_means <- colMeans(phi)
cat("Average acceptance rate with c = 1: ", mean(as.vector(metro2$acc.prob)))
cat("\n Average acceptance rate with c = 0.5: ", avg_acc)
par(mfrow=c(2,2))
for(i in 1:ncol(phi)){
  plot(phi[,i], type="l", xlab="Iteration", ylab=paste0("phi_", i), main=paste0("phi_", i))
}
par(mfrow=c(2,2))
for(i in 1:ncol(phi)){
  hist(phi[,i], freq=FALSE, xlab=paste0("phi_", i), main=paste0("phi_", i))
  #lines(density(phi[,i]), col="red")
}
par(mfrow=c(2,2))
hist(phi[,1], freq = FALSE, main="phi_1 c)", xlab = "phi_1")
hist(phi_b[,1], freq = FALSE, main="phi_1 b)", xlab = "phi_1")
hist(phi[,3], freq = FALSE, main="phi_3 c)", xlab = "phi_3")
hist(phi_b[,3], freq = FALSE, main="phi_3 b)", xlab = "phi_3")
par(mfrow=c(2,2))
hist(phi[,6], freq = FALSE, main="phi_6 c)", xlab = "phi_6")
hist(phi_b[,6], freq = FALSE, main="phi_6 b)", xlab = "phi_6")
hist(phi[,9], freq = FALSE, main="phi_9 c)", xlab = "phi_9")
hist(phi_b[,9], freq = FALSE, main="phi_9 b)", xlab = "phi_9")
x_vec <- c(1,1,1,1,0,0,0,1,0.5)
lambda <- exp(x_vec%*%t(beta_metro))
y_mat <- apply(lambda, 1, function(i){rpois(1000, i)})
hist(y_mat, freq = FALSE, xlab="y_pred", main="Predictive Distribution of nBids",breaks = 25)
prob_y0 <- length(y_mat[y_mat==0])/length(y_mat)
cat("Probability of nBids=0: ", prob_y0)
```