Bayesian Learning

Computer Lab 4

Elvin Granat (elvgr805)
Max Fischer (maxfi539)

1) Poisson regression, the MCMC way.

a)

To obtain the maximum likelihood estimator of β in the Poisson regression model for the eBay dataset, we initially imported the data from the .dat-file.

```
### Setup
data.ebay = read.table("eBayNumberOfBidderData.dat", header=TRUE)
```

A Poisson regression model was then fitted to the data, with nBids as ground truth and the rest of the data as covariates.

```
# a)
# Fit the model
glm.fit = glm(nBids ~ .-Const, data=data.ebay ,family="poisson")

# Check significance
summary(glm.fit)
```

Significance of variables:

Covariate	Estimate	Std	z-value	Significance
(Intercept)	1.07244	0.03077	34.848	Pr(>|z|) < 2e-16
PowerSeller	-0.02054	0.03678	-0.558	Pr(>|z|) = 0.5765
VerifyID	-0.39452	0.09243	-4.268	Pr(>|z|) = 1.97e-05
Sealed	0.44384	0.05056	8.778	Pr(>|z|) < 2e-16
MinBlem	-0.05220	0.06020	-0.867	Pr(>|z|) = 0.3859
MajBlem	-0.22087	0.09144	-2.416	Pr(>|z|) = 0.0157
LargNeg	0.07067	0.05633	1.255	Pr(>|z|) = 0.2096
LogBook	-0.12068	0.02896	-4.166	Pr(>|z|) = 3.09e-05
MinBidShare	-1.89410	0.07124	-26.588	Pr(>|z|) < 2e-16

From the summary above, we can draw the conclusion that the following covariates' confidence interval do not include zero (with the probability written afterwards) and thus are significant:

- (Intercept) (99,9%)
- VerifyID (99,9%)
- Sealed (99,9%)
- LogBook (99,9%)
- MinBidShare (99,9%)
- MajBlem (95%)

b)

A Bayesian analysis of the Poisson regression was done.

- Zellner g-prior: $\beta \sim N(0,\ 100 * (X^T X)^{-1}$ (X is the n x p covariate matrix)
- Likelihood: $y_i \mid \beta \sim Poisson[\ exp(x_i\beta)\ ]$, i = 1,...,n
- Posterior: $\beta \mid y_i \sim N[\ \widehat{\beta},\ J_y^{-1}(\widehat{\beta})\ ]$, $\widehat{\beta}$: Mode of β, $J_y^{-1}(\widehat{\beta})$: Inverse negative Hessian at the mode of β.

To calculate the mode of β and the inverse negative Hessian at the mode of β a function calculating the log posterior was coded:

```
logPostPoisson = function(β, mu0, Sigma0, X, y) {
  p = length(β)

  # log of the likelihood
  log.likelihood = sum(y * (X %*% β) - exp(X %*% β))

  # if likelihood is very large or very small, stear optim away
  if (abs(log.likelihood) == Inf) log.likelihood = -20000;

  # log of the prior
  log.prior = dmvnorm(β, mean = mu0, sigma = Sigma0, log = TRUE)

  return(log.likelihood + log.prior)
}
```

The likelihood function was derived in the following way:

$$L(y \mid \beta) \ = \ \prod_{i=1}^{n} p(y_i \mid \beta) = \prod_{i=1}^{n} \frac{(e^{x_i\beta})^{y_i}}{y_i!} e^{-(e^{x_i\beta})} = \prod_{i=1}^{n} \frac{e^{y_i x_i \beta}}{y_i!} e^{-(e^{x_i\beta})}$$

By using the likelihood function above, the log-likelihood function was derived:

$$l(y \mid \beta) \ = \ log[\prod_{i=1}^{n} \frac{e^{y_i x_i \beta}}{y_i!} e^{-(e^{x_i\beta})}] \ = \ log[\prod_{i=1}^{n} e^{y_i x_i \beta}] + log[\prod_{i=1}^{n} e^{-(e^{x_i\beta})}] - log[\prod_{i=1}^{n} y_i!] \ =$$

$$= \ log[e^{\sum_{i=1}^{n} y_i x_i \beta}] + log[e^{\sum_{i=1}^{n} -(e^{x_i\beta})}] - log[\prod_{i=1}^{n} y_i!] \propto \ \sum_{i=1}^{n} y_i x_i \beta * log[e] + \sum_{i=1}^{n} - (e^{x_i\beta}) * log[e]$$

$$= \ \sum_{i=1}^{n} (y_i x_i \beta \ - \ e^{x_i\beta})$$

The log-posterior function was numerically optimized by using the optim.R package:

```
y.ebay = data.ebay[,1]
X.ebay = as.matrix(data.ebay[,-1])
# Prior
β0 = rep(0, 9) # mean prior
Sigma0 = 100 * solve(t(X.ebay) %*% X.ebay) # Sigma prior

# Optimize posterior of β given priors and data
optim.res = optim(β0, logPostPoisson, gr=NULL,
                  β0, Sigma0, X.ebay, y.ebay,
                  method="BFGS", control=list(fnscale=-1), hessian=TRUE)

# Results
β.mode = optim.res$par
β.invhessian = -solve(optim.res$hessian)
```

Mode of beta:

	β_1	β_2	β_3	β_4	β_5	β_6	β_7	β_8	β_9
Mode	1.069841	-0.020512	-0.393006	0.443556	-0.052466	-0.221238	0.070697	-0.120218	-1.891985

Inverse negative Hessian:

	β_1	β_2	β_3	β_4	β_5	β_6	β_7	β_8	β_9
β_1	9.454625e-04	-7.138972e-04	-2.741517e-04	-2.709016e-04	-4.454554e-04	-2.772239e-04	-5.128351e-04	6.436765e-05	1.109935e-03
β_2	-7.138972e-04	1.353076e-03	4.024623e-05	-2.948968e-04	1.142960e-04	-2.082668e-04	2.801777e-04	1.181852e-04	-5.685706e-04
β_3	-2.741517e-04	4.024623e-05	8.515360e-03	-7.824886e-04	-1.013613e-04	2.282539e-04	3.313568e-04	-3.191869e-04	-4.292827e-04
β_{41}	-2.709016e-04	-2.948968e-04	-7.824886e-04	2.557778e-03	3.577158e-04	4.532308e-04	3.376467e-04	-1.311025e-04	-5.759169e-05
β_5	-4.454554e-04	1.142960e-04	-1.013613e-04	3.577158e-04	3.624606e-03	3.492353e-04	5.844006e-05	5.854011e-05	-6.437066e-05
β_6	-2.772239e-04	-2.082668e-04	2.282539e-04	4.532308e-04	3.492353e-04	8.365059e-03	4.048644e-04	-8.975843e-05	2.622264e-04
β_7	-5.128351e-04	2.801777e-04	3.313568e-04	3.376467e-04	5.844006e-05	4.048644e-04	3.175060e-03	-2.541751e-04	-1.063169e-04
β_8	6.436765e-05	1.181852e-04	-3.191869e-04	-1.311025e-04	5.854011e-05	-8.975843e-05	-2.541751e-04	8.384703e-04	1.037428e-03
β_9	1.109935e-03	-5.685706e-04	-4.292827e-04	-5.759169e-05	-6.437066e-05	2.622264e-04	-1.063169e-04	1.037428e-03	5.054757e-03

c) We now simulate from the actual posterior of β using Metropolis algorithm. To do this we program a general function that uses the Metropolis algorithm from any posterior distribution.

The proposal density, given the usage of random walk Metropolis is:

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

Where:

$$\Sigma = J_{\mathbf{y}}^{-1}(\tilde{\beta})$$

(Inverse negative hessian from b))

And c is a given tuning parameter.

```
Metropolis = function(nBurnIn, nSamples, theta, c, logPostFunc, ...) {
```

The Metropolis function takes in a number of parameters:

nBurnIn = number of burnin iterations of the algorithm
nSamples = number of samples after the burnin iterations
theta = the parameter of which to sample, used to evaluate the posterior density
c = tuning parameter for the proposal density
logPostFunc = computes the log posterior density of the parameters
… = prior hyperparameters used in logPostFunc together with theta to calculate the log posterior density

Final Metropolis function is as follows:

```
Metropolis = function(nBurnIn, nSamples, theta, c, logPostFunc, ...) {
  # Setup
  theta.c = theta
  Sigma.c = c * β.invhessian
  nAccepted = 0
  p = length(theta)
  theta.samples = matrix(NA, nSamples, p)

  # Iterations
  for(i in -nBurnIn : nSamples) {
    # Sample from proposal distribution
    theta.p = as.vector(rmvnorm(1, mean = theta.c, sigma = Sigma.c))

    # Calculate log posteriors
    log.post.p = logPostFunc(theta.p, ...)
    log.post.c = logPostFunc(theta.c, ...)

    # Calculate alpha
    alpha = min(1, exp(log.post.p - log.post.c))

    # Select sample with probability alpha
    u = runif(1)
    if (u <= alpha){
      theta.c = theta.p
      nAccepted = nAccepted + 1
    }

    # Save sample if not burnin
    if (i>0) theta.samples[i,] = theta.c
  }
  cat("Sampled", nSamples, "samples with an acceptance rate of", nAccepted/nSamples)
  return(theta.samples)
}
```

Using number of burnin iterations, number of samples and tuning parameter c of:

```
# Setup sampling
c = 0.5
nSamples = 4000
nBurnIn = 1000
```

And calling the function as follows:

```
# Samples from posterior using metropolis
β.samples = Metropolis(nBurnIn, nSamples, β.mode, c, logPostPoisson, β0, Sigma0, X.ebay, y.ebay)
```

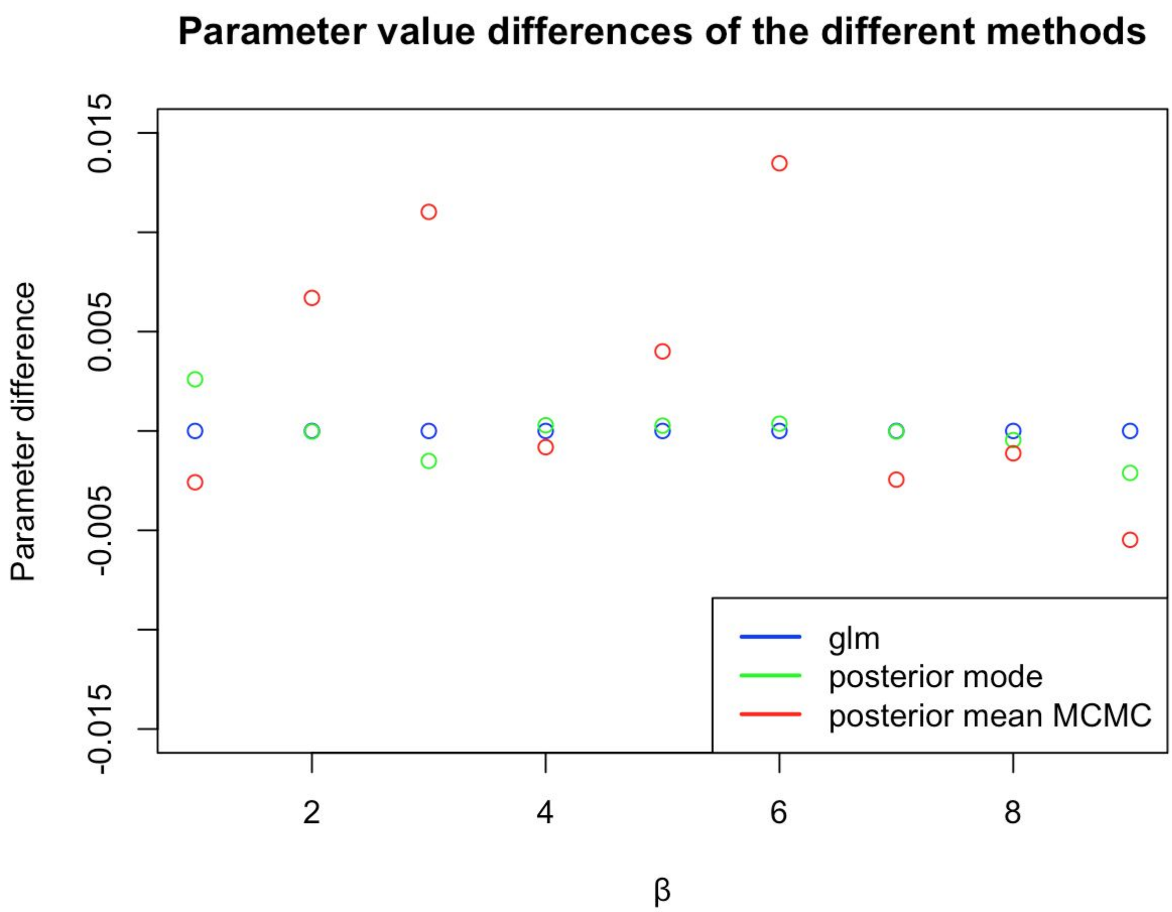
Results:

```
# Estimate parameters using posterior mean
β.post.mean = apply(β.samples, 2, mean)
```

Mean of beta:

	β_1	β_2	β_3	β_4	β_5	β_6	β_7	β_8	β_9
Mean	1.075024	-0.027237	-0.405541	0.444659	-0.056203	-0.234342	0.073121	-0.119552	-1.888617

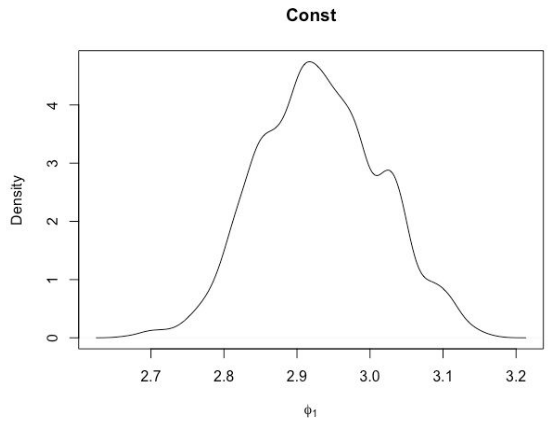
To graphically compare the parameter values of the different estimation methods we plot the value difference of the parameters. With the result of glm method as the middle values. Results in the following plot:

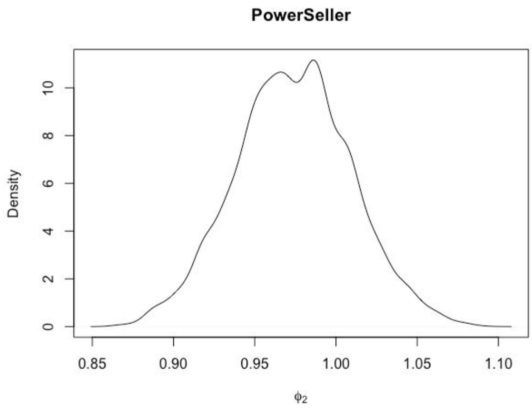


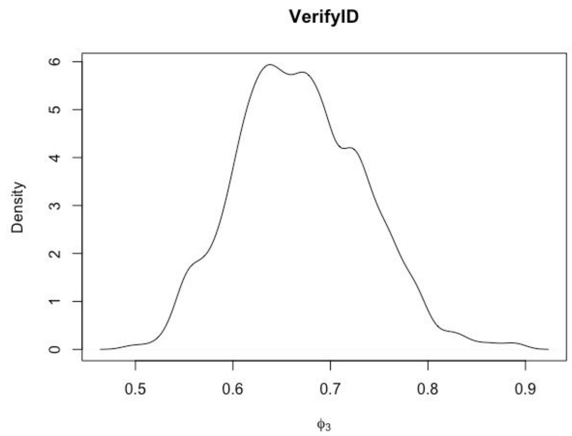
We can see that glm and posterior mode using optim reults in similar results whereas MCMC gives very different parameter values for some paramaters.

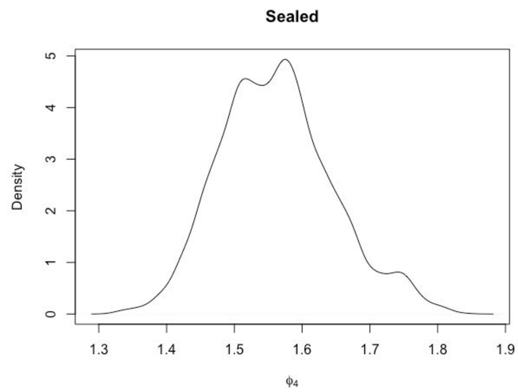
To visualize the convergence MCMC we compute the posterior distribution of:

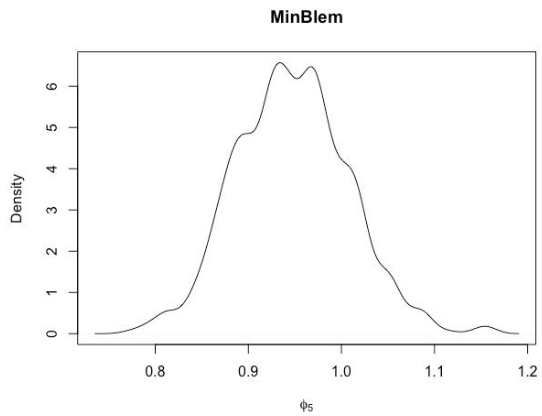
$\phi_j = \exp(\beta_j)$ for the sampled betas.

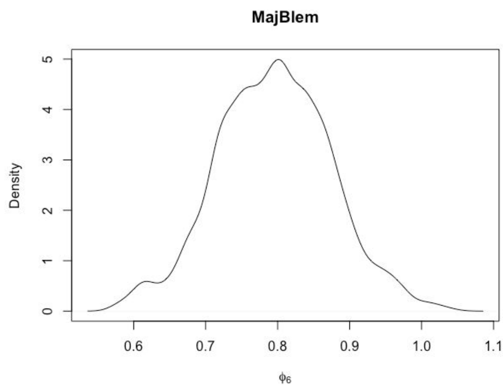


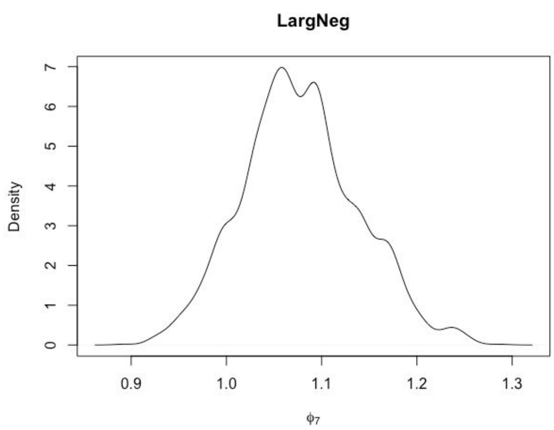


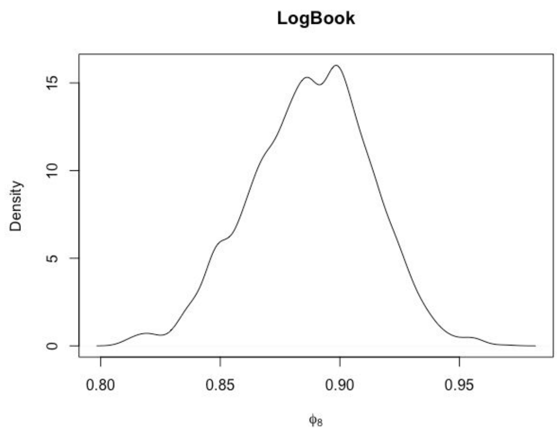


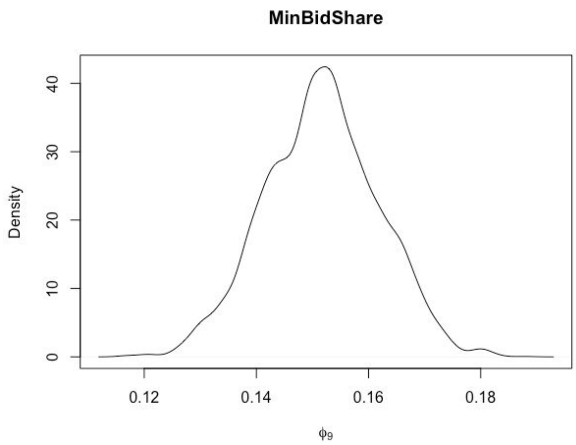












Which looks like reasonable posterior distributions. Interesting that we can see that the range of the parameter values that were deemed significant in a) don't cross 1 with any of the samples. Whereas the not so significant covariates' parameters have most of its mass close to 1.

d)
We now use the draws from c) to calculate the probability that a certain auction will have 0 bids.

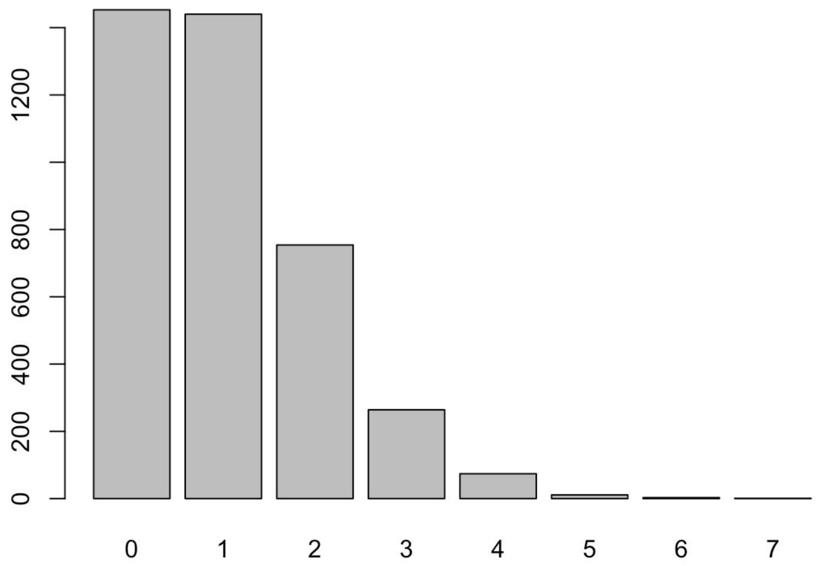
Given auction:

- $\mathbf{PowerSeller} = 1$
- $\mathbf{VerifyID} = 1$
- $\mathbf{Sealed} = 1$
- $\mathbf{MinBlem} = 0$
- $\mathbf{MajBlem} = 0$
- $\mathbf{LargNeg} = 0$
- $\mathbf{LogBook} = 1$
- $\mathbf{MinBidShare} = 0.5$

Where we calculate the probability using samples from the poisson distribtion with lambda value calculated using the samples from the MCMC. And finally calculating the probability of having 0 bids using the proportion of the samples that had 0 bids.

```
# Auction vector
x = c(1, 1, 1, 1, 0, 0, 0, 1, 0.5)

# Setup for sampling
nBids.samples = numeric()

# Sample nBids
for(i in 1:nSamples) {
  nBids.samples[i] = rpois(1, exp(x %*% β.samples[i,]))
}

# Plot distribution
barplot(table(nBids.samples))

# Calculate probabily of no bidders
prob = sum(nBids.samples == 0) / nSamples # 0.36
```

With corresponding distribution:



Resulting in a **36%** probability of 0 bids.