# Bayesian Learning

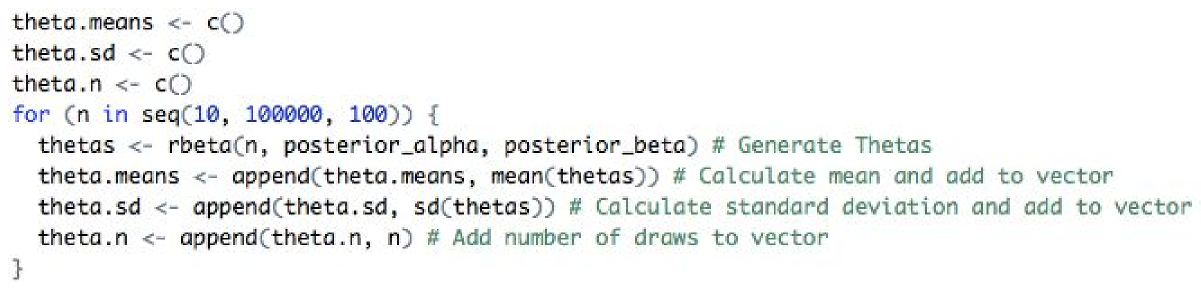
Computer Lab 1

## 1. Bernoulli … again

We assume that y1, ..., yn|θ ~ Bern(θ), and that we have obtained a sample with s = 14 successes in n = 20 trials. Assume a Beta( $\alpha_0$ , $\beta_0$ ) prior for θ and let $\alpha_0$ = $\beta_0$ = 2.
f = n - s = 20 - 14 = 6

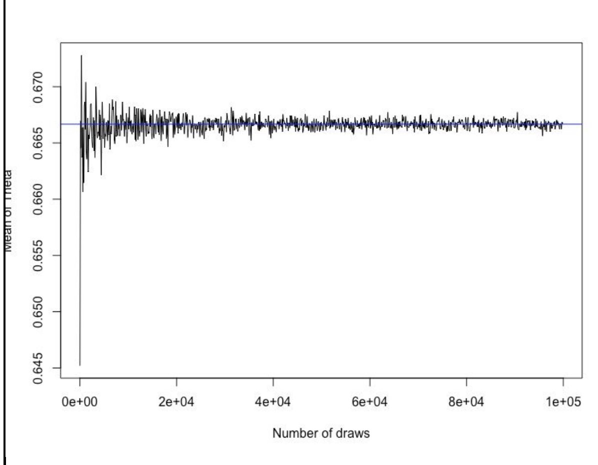
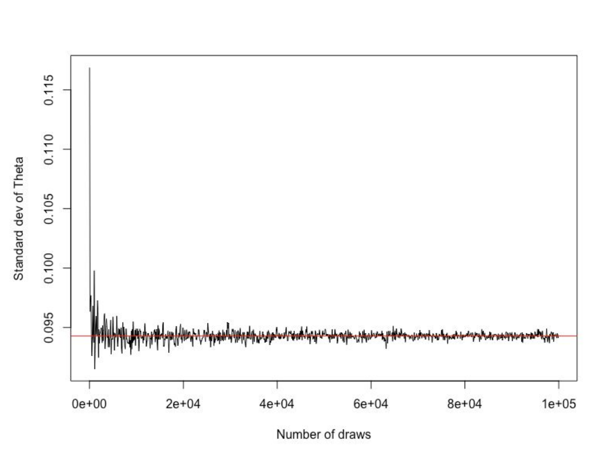
a)
posterior_alpha: 2 + 14 = 16
posterior_beta: 2 + 6 = 8
True posterior mean = $\frac{16}{16+8} = \frac{16}{24} = 0,666667$
True posterior standard deviation = $\sqrt{\frac{16*8}{(16+8)^2*(16+8+1)}} = 0.09428$

To verify that the mean and standard deviation converges towards the true values as the number of draws grows large, different sized draws (nDraws = 10, 20, … , 100000) from the posterior distribution $\theta\,|\,y \sim Beta\,(\alpha_0$ + s, $\beta_0$ + f ) was done. The mean and standard deviation was then calculated for each draw and plotted on two separate plots. The plots verify that as the number of draws grows large, the mean and standard deviation converges towards the true values.



```
theta.means <- c()
theta.sd <- c()
theta.n <- c()
for (n in seq(10, 100000, 100)) {
  thetas <- rbeta(n, posterior_alpha, posterior_beta) # Generate Thetas
  theta.means <- append(theta.means, mean(thetas)) # Calculate mean and add to vector
  theta.sd <- append(theta.sd, sd(thetas)) # Calculate standard deviation and add to vector
  theta.n <- append(theta.n, n) # Add number of draws to vector
}
```

**Graphical representation of mean and standard deviation of theta.**

| | |
|---|---|
|  |  |
| *Mean of thetas, as the number of draws increases* | *Standard deviation of theta, as the number of draws increases* |

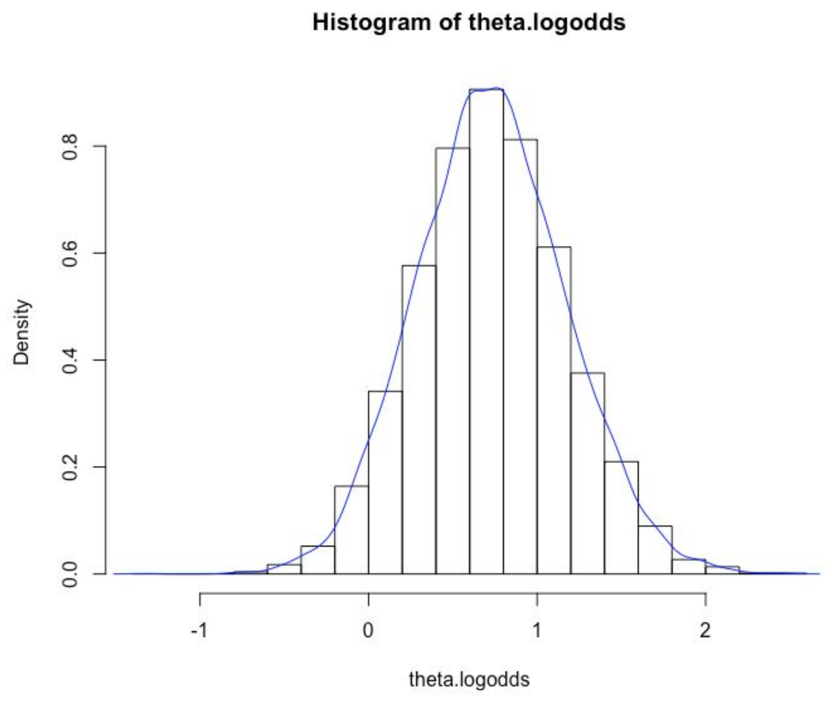
b)
To compute the posterior probability Pr( $\theta < 0.4 \mid y$ ) by simulation, we start of by simulating 10000 draws from the beta-distribution, posterior_alpha = 16, posterior_beta = 8. By dividing the number of simulated theta-values below 0.4 with the total number of simulated theta-values (10000), the posterior probability is obtained.

Simulated posterior probability: 0.0045
Actual posterior probability: 0.003972681

```
theta.draws <- rbeta(10000, posterior_alpha, posterior_beta) # Draw 10000 numbers
theta.no_smaller <- ifelse(theta.draws < 0.4, 1, 0) # If draw < 0.4 -> 1, else -> 0
theta.prob_smaller <- sum(theta.no_smaller)/length(theta.draws) # Calculate probability of drawn number being < 0.4
print(theta.prob_smaller)
print(pbeta(0.4,posterior_alpha, posterior_beta)) # Actual probability of b
```

c)
To compute the posterior probability of the log-odds φ = log( $\frac{\theta}{1-\theta}$ ) we simply use the 10000 draws done in task b and insert them in the log-odds function. The result was then plotted on a histogram. The density function was then plotted on top of the histogram.

```
theta.logodds <- log(theta.draws/(1-theta.draws))
theta.logodds.density <- density(theta.logodds)
hist(theta.logodds, probability = TRUE)
lines(theta.logodds.density, col='blue')
```



## 2. Log-normal distrubtion and the Gini coefficient

Given observations y of salaries modelled as *logN*(μ, $\sigma^2$) and density function:

$$p(y|\mu,\sigma^2) = \frac{1}{y\cdot\sqrt{2\pi\sigma^2}}\exp\left[-\frac{1}{2\sigma^2}(\log y-\mu)^2\right]$$

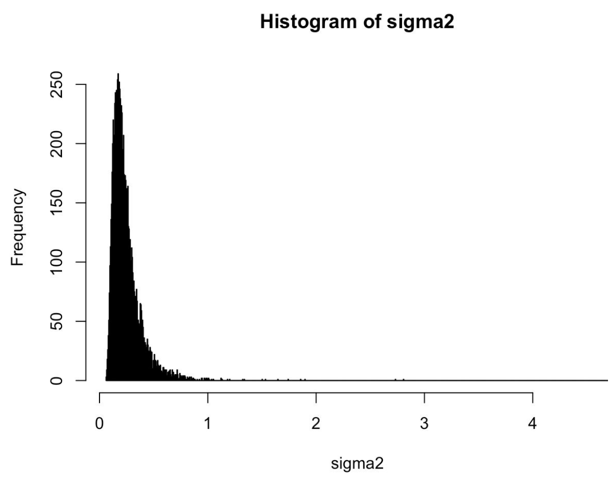
μ is assumed to be known = 3.5

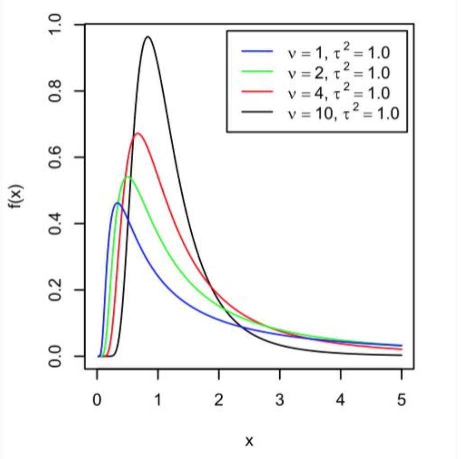
$\sigma^2$ is unknown with a non-informative prior $p(\sigma^2) = 1/\sigma^2$, resulting in the posterior for $\sigma^2$ is the

Scaled Inv-$\chi^2$ distribution $Inv-\chi^2(n,\tau^2)$ where $\tau^2 = \frac{\sum_{i=1}^{n}(\log y_i-\mu)^2}{n}$.

**a)** Simulate 10 000 draws from the posterior $\sigma^2$:

```
tau2 <- function(..Y, my) {
  n <- length(..Y)
  return(sum((log(..Y)-my)^2)/n)
}

data <- c(14, 25, 45, 25, 30, 33, 19, 50, 34, 67)
my <- 3.5
n <- length(data)
nDraws = 10000

draw <- rchisq(n=nDraws, df=n)
sigma2 <- n*tau2(data, my)/draw
hist(sigma2, breaks=1000)
```



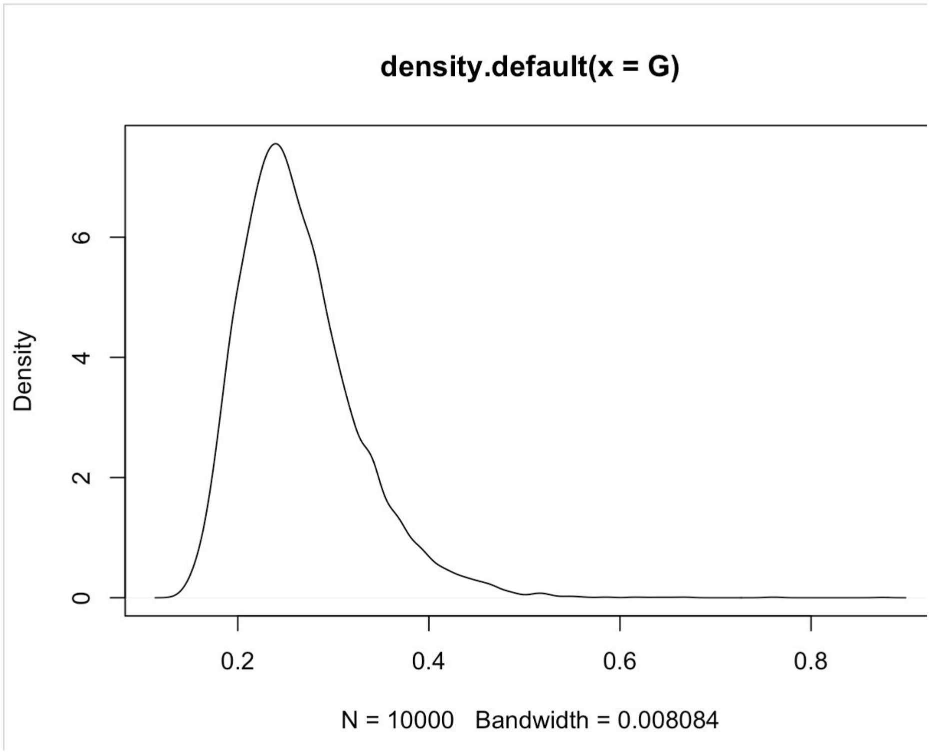


Compared to the theoretical $Inv-\chi^2(n,\tau^2)$ taken fromwikipedia it has a similar shape.

**b)** Using Gini coefficient to measure income inequality. Given that incomes follow a logN distribution then $G = 2\Phi\left(\sigma/\sqrt{2}\right) - 1$ where Φ(σ/sqrt(2)) is the probability of x <= σ/sqrt(2) in a N(0,1) distribution.

Using the posterior draws from a) we compute the posterior distribution of G for current data:

```
sigma = sqrt(sigma2)
G <- 2 * pnorm(sigma/sqrt(2), mean = 0, sd = 1) - 1
hist(G, breaks=100)
```

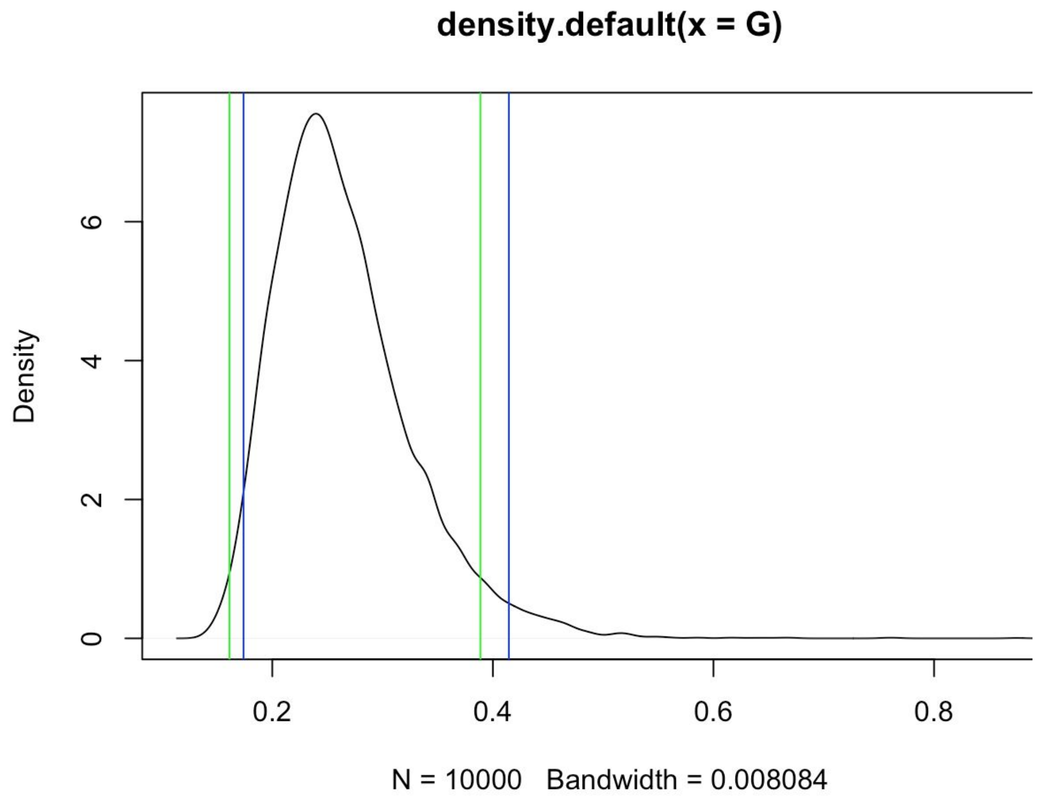


**c)** Computing the 95% equal tail credible interval for G using the posterior draws from b)

```
# equal tail
G.sorted <- sort(G)[(nDraws*0.025+1):(nDraws*0.975)]
G.credible_interval <- c(min(G.sorted), max(G.sorted)) # Credible interval
print(G.credible_interval) #0.1738928 0.4144967
```

Compute 95% highest posterior density interval:

```
# density
G.density <- density(G)
G.density.df <- data.frame(x = G.density$x, y = G.density$y)

G.density.ordered <- G.density.df[with(G.density.df, order(y)),]
G.density.ordered <- data.frame(x = G.density.ordered$x, y = G.density.ordered$y)
G.density.dn = cumsum(G.density.ordered$y)/sum(G.density.ordered$y)
G.0.05 = which(G.density.dn >= 0.05)[1]
G.density.ordered.95 = G.density.ordered[(G.0.05+1):512,]
G.density.interval <- c(min(G.density.ordered.95$x),max(G.density.ordered.95$x))
```



Where green depicts the HPD interval and blue depicts the credible interval. We can see that the HPD interval is closer to the mass of the posterior which seems like a more reasonable estimated interval result for G.

## 3. Bayeisan inference - von Mises distribution

**a)** Given observations y of modelled as a von Mises distribution and a exponential prior knowledge of the value of kappa):

(note: k = kappa)

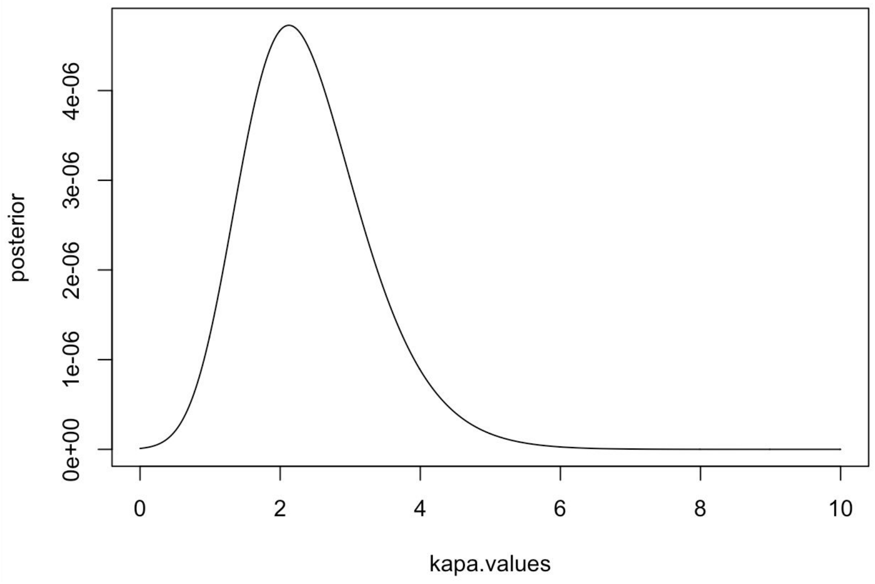
$$Prior:\ p(k) = \lambda e^{-\lambda k} = //\lambda = 1// = e^{-k}$$

$$Likelihood:\ p(Y \mid k,\ \mu) = p(\ Y \mid k) = //\ y\ independent\ // = \prod_{i=1}^{10} p(y \mid k\ ) = \prod_{i=1}^{10} \frac{exp\{k * cos\ (\ y - \mu)\}}{2\pi I_0(k)}$$

$$Posterior:\ p(k|Y,\mu) = //\ \mu\ known\ // = p(k \mid Y\ ) = p(Y \mid k)\ *\ p\ (k) = \prod_{i=1}^{10} \frac{exp\{k * cos\ (\ y - \mu)\}}{2\pi I_0(k)}\ *\ e^{-k}$$

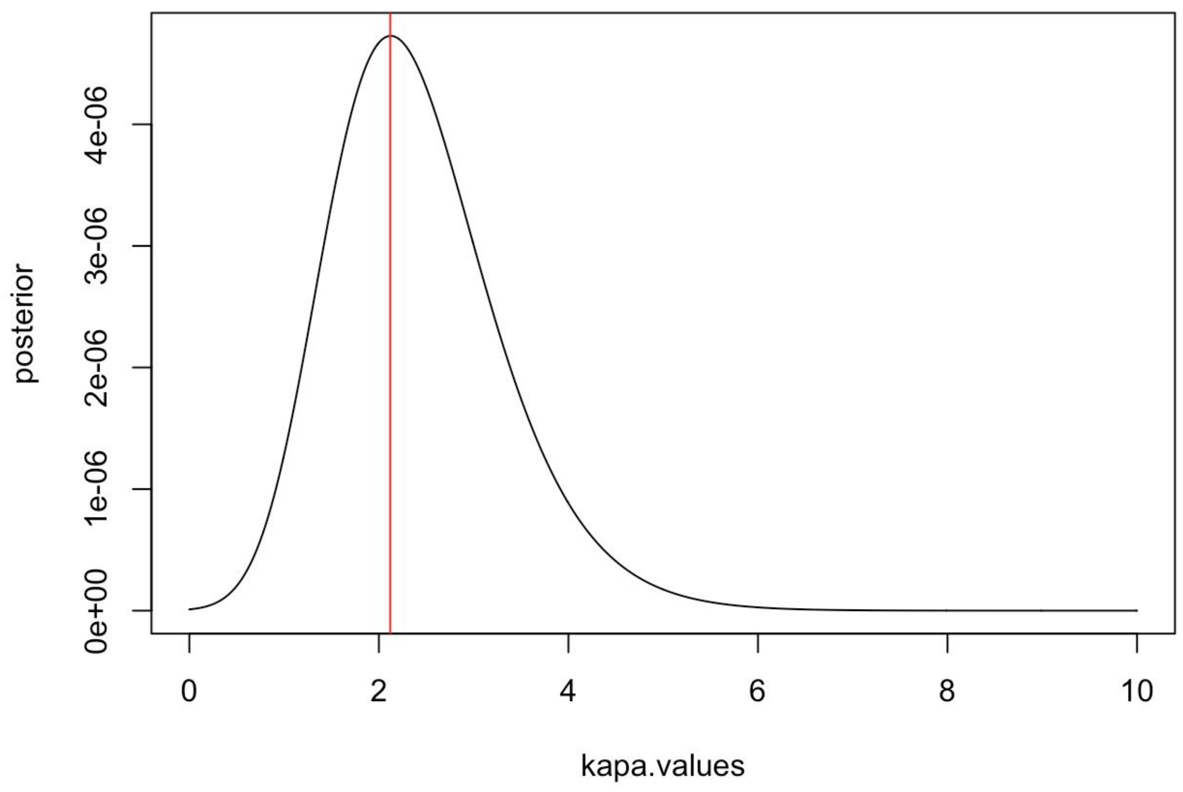
Plotting the posterior of kappa using a fine grid of kappa values:

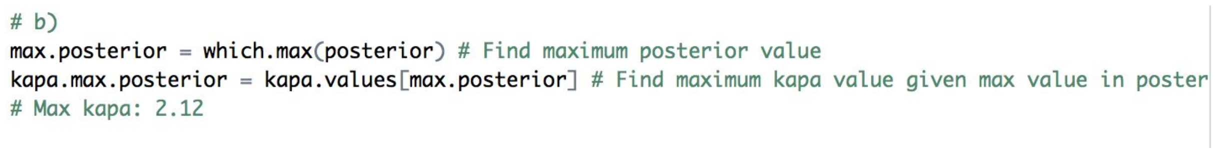
```r
likelihoodVonMises <- function(y, k, my) {
    return(prod(exp(k*cos(y-my))/(2*pi*besselI(k, nu=0))))
}

y.values <- c(-2.44, 2.14, 2.54, 1.83, 2.02, 2.33, -2.79, 2.23, 2.07, 2.02)
kapa.values <- seq(0, 10, 0.01)
my <- 2.39

# Vector of prior values, given kapas
kapa.prior <- dexp(kapa.values)

# Calculate likelihood for each kapa, given y-vector
likelihoods = numeric()
i = 1
for(k in kapa.values) {
  likelihoods[i] = likelihoodVonMises(y.values, k, my)
  i = i + 1
}

posterior = likelihoods * kapa.prior # Vector of posterior values
plot(kapa.values, posterior, type='l') # Plot
```



**b)**

Observing the graph from a) we find the approximate posterior mode of kappa taking the kappa resulting in the highest posterior probability for given y and kappa. Reading from the image around 2.10 seems reasonable:





```
# b)
max.posterior = which.max(posterior) # Find maximum posterior value
kapa.max.posterior = kapa.values[max.posterior] # Find maximum kapa value given max value in poster
# Max kapa: 2.12
```

Calculated from the grid values gives kappa = 2.12

# Bayesian Learning

Computer Lab 2

# 1. Linear and polynomial regression

A dataset containing daily temperatures (in celsius degree) at Malmslätt, Linköping, was used to perform a Bayesian analysis of the following quadratic regression:

$$temp = \beta_0 + \beta_1 * time + \beta_2 * time^2 + \varepsilon, \ \varepsilon \sim N(0, \sigma^2)$$

where the response variable is time and the covariate is

$$time = \frac{the\ number\ of\ days\ since\ beginning\ of\ year}{366}$$

a) The following priors to the linear regression was used:

- $\beta|\sigma^2 \sim N(\mu_o, \sigma^2\Omega_o^{-1})$
- $\sigma^2 \sim Inv - \chi^2(v_o, \sigma_o^2)$

$\mu_0$ (the beta-values) was chosen as follows:

- $\beta_0 = -5$
- $\beta_1 = 20$
- $\beta_2 = 60$

$\beta_0$ was set to -5 as it seems reasonable that the temperature at time = 0 (January 1st) is around -5 degrees.
$\beta_1$ and $\beta_2$ was chosen to generate a linear regression model that estimate the temperature for 30th of June to 20 degrees (time = 0.5).

$v_0$ is representing the degrees of freedom in the prior, thus how sure we are about our prior knowledge. The initial value for $v_0$ was set to 60.

It seems possible that the temperature will differ with 7 degrees in 95 % of the cases. $\sigma_o^2$ was set to:

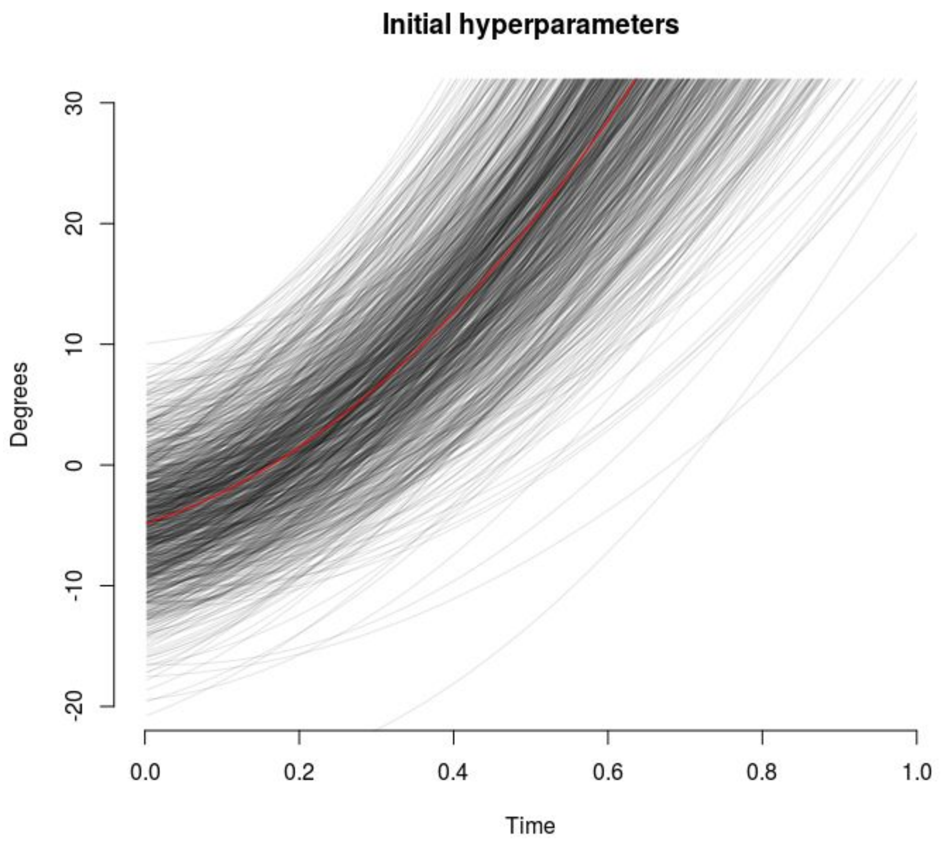
- $\sigma_o^2 = (\frac{7}{1.96})^2 = 12.5771$

$\Omega_0$ is a vector describing how sure we are about our betas. As we are more certain about $\beta_0$ than $\beta_1$ and $\beta_2$ , $\Omega_0$ was set to:

- $\Omega_0$ = (0.5, 0.1, 0.1)

b)
To find out if our chosen hyper parameters and priors are sensible, 1000 draws of beta was done from the joint prior of all parameters. A regression curve was then computed and plotted for each draw. A read mean-curve of all regression lines is visible in the middle of the plot.

```
# simulation
chi2 = rchisq(v0, n=nDraws)
sigma2_draws = v0*sigma2_0/chi2
omega0_inv = solve(omega0)
B_draws = matrix(0, nDraws, 3)

# start new plot
plot.new()
plot.window(xlim=c(0,1), ylim=c(-20,30))
axis(side=1)
axis(side=2)

# simulate draws
for(i in 1:nDraws){
  B_draws[i,] = mvrnorm(n = 1, mu = my0, Sigma=sigma2_draws[i]*omega0_inv)
  lines(data$time, B_draws[i,1]+B_draws[i,2]*data$time+B_draws[i,3]*data$time^2, col=rgb(0,0,0,0.1))
}
lines(data$time, mean(B_draws[,1])+mean(B_draws[,2])*data$time+mean(B_draws[,3])*data$time^2, col=rgb(1,0,0,1))
title(main="Initial hyperparameters", xlab="Time", ylab="Degrees")
```

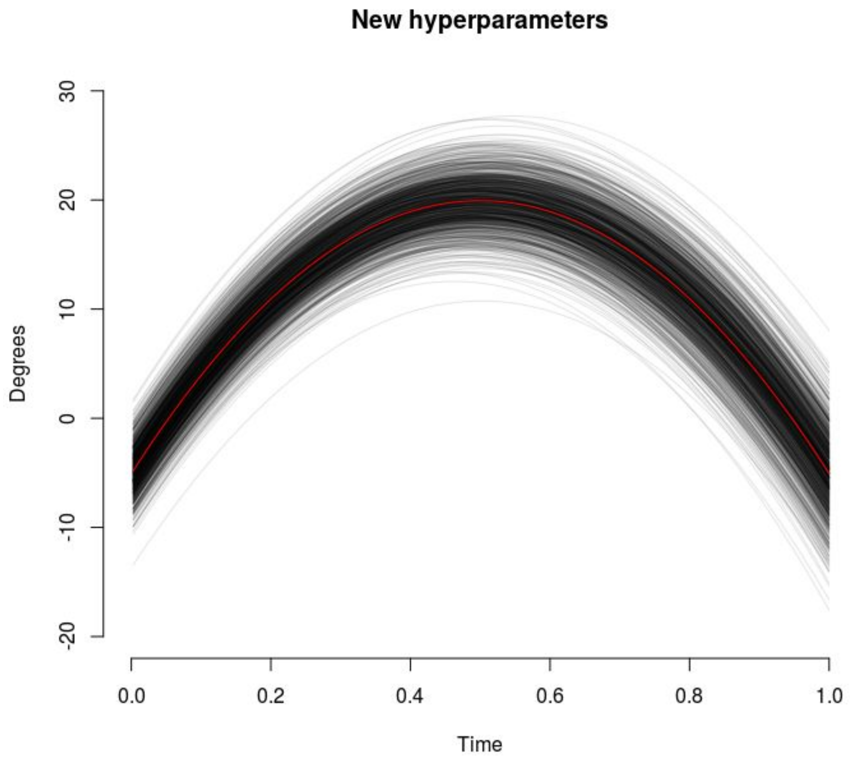


One reasonable pattern that should appear is that the regression lines should be at almost the same degree at time = 0 and time = 1, which is not the case in the curve above. It's also seems like the regression lines are exponentially increasing, a pattern that doesn't represent the temperature during a year. Thus the chosen hyper parameters are not sensible.

New hyper parameters was chosen:

- $\beta_0 = -5$
- $\beta_1 = 100$
- $\beta_2 = -100$
- $\sigma_o^2$ = 2
- 0= (0.5, 0.5, 0.5)

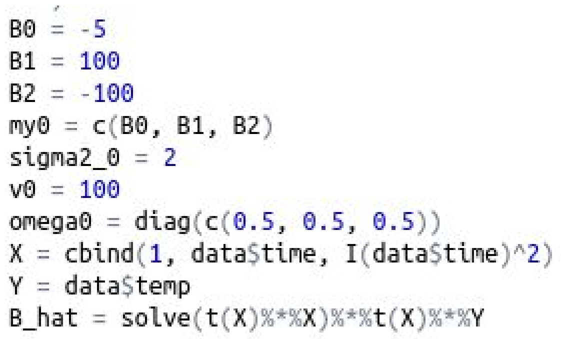
New draws of beta was done and regression lines was computed and plotted:



The plot above has the same temperature at time = 0 and time = 1. It also has the lowest temperature in the beginning/end of the year, it increases during the spring, peaks in the middle of the summer end dips towards the fall. The newly chosen hyper parameters seams reasonable.

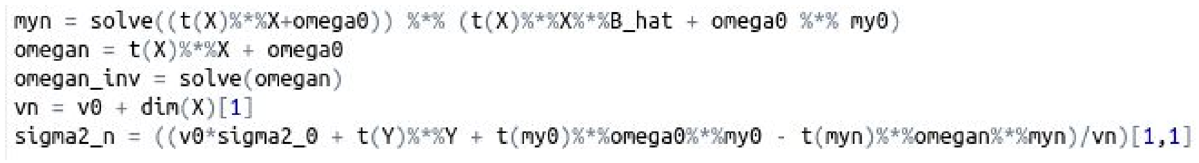
c)

The same hyper parameters as in b) was used. To calculate $\mu_n$ , beta-hat was calculated.



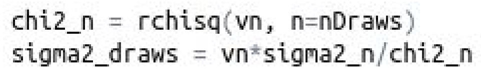
```
B0 = -5
B1 = 100
B2 = -100
my0 = c(B0, B1, B2)
sigma2_0 = 2
v0 = 100
omega0 = diag(c(0.5, 0.5, 0.5))
X = cbind(1, data$time, I(data$time)^2)
Y = data$temp
B_hat = solve(t(X)%*%X)%*%t(X)%*%Y
```

The rest of the parameters needed to calculate the posterior was calculated:



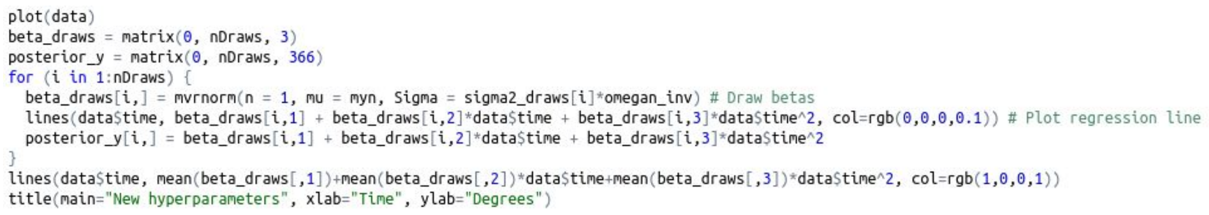
```
myn = solve((t(X)%*%X+omega0)) %*% (t(X)%*%X%*%B_hat + omega0 %*% my0)
omegan = t(X)%*%X + omega0
omegan_inv = solve(omegan)
vn = v0 + dim(X)[1]
sigma2_n = ((v0*sigma2_0 + t(Y)%*%Y + t(my0)%*%omega0%*%my0 - t(myn)%*%omegan%*%myn)/vn)[1,1]
```

The posterior $\sigma^2$ was drawn as before:



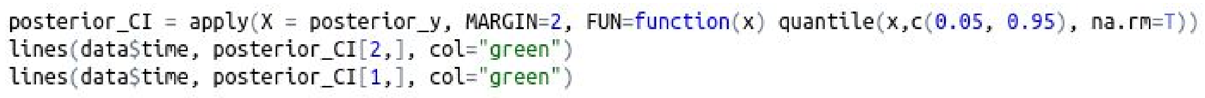
```
chi2_n = rchisq(vn, n=nDraws)
sigma2_draws = vn*sigma2_n/chi2_n
```

The data was then plotted and nDraws of beta was done. For each draw of beta, a regression line was calculated and plotted.

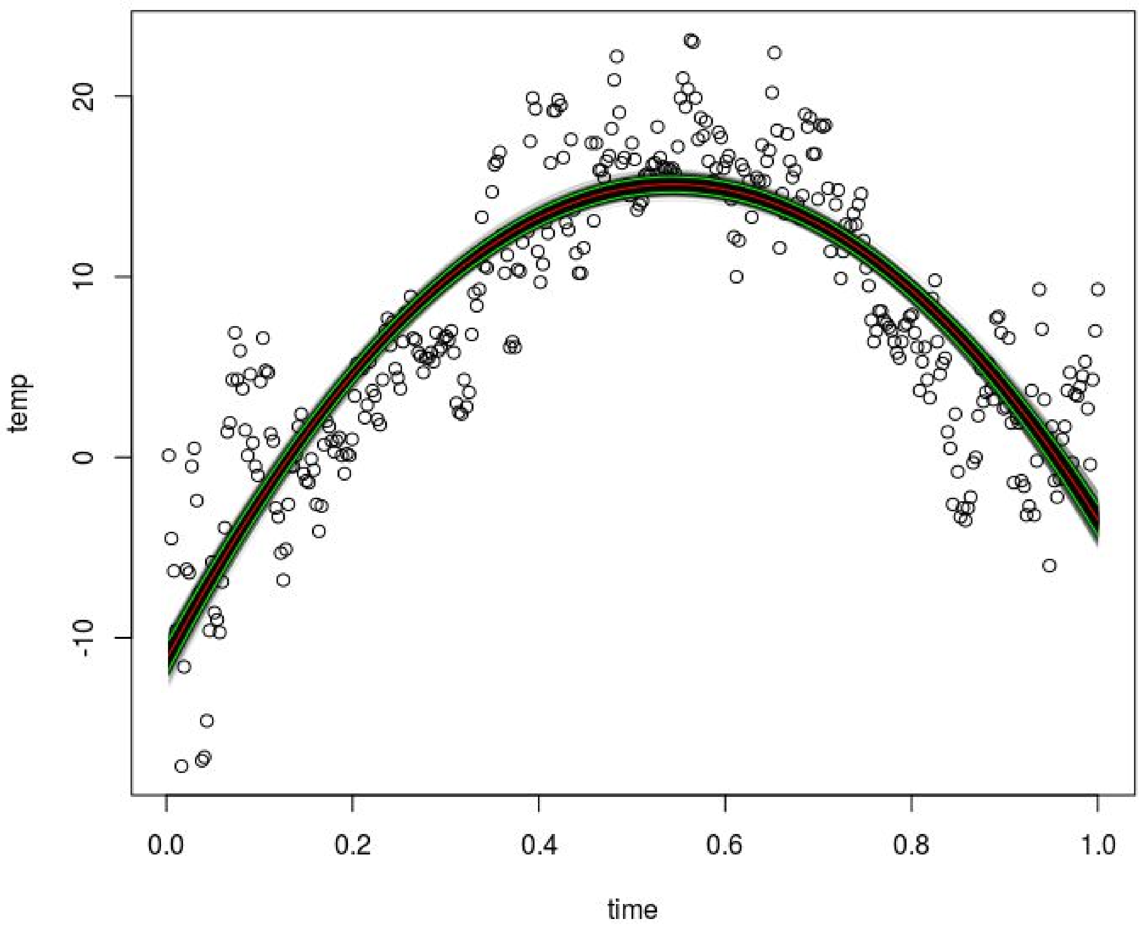


```
plot(data)
beta_draws = matrix(0, nDraws, 3)
posterior_y = matrix(0, nDraws, 366)
for (i in 1:nDraws) {
  beta_draws[i,] = mvrnorm(n = 1, mu = myn, Sigma = sigma2_draws[i]*omegan_inv) # Draw betas
  lines(data$time, beta_draws[i,1] + beta_draws[i,2]*data$time + beta_draws[i,3]*data$time^2, col=rgb(0,0,0,0.1)) # Plot regression line
  posterior_y[i,] = beta_draws[i,1] + beta_draws[i,2]*data$time + beta_draws[i,3]*data$time^2
}
lines(data$time, mean(beta_draws[,1])+mean(beta_draws[,2])*data$time+mean(beta_draws[,3])*data$time^2, col=rgb(1,0,0,1))
title(main="New hyperparameters", xlab="Time", ylab="Degrees")
```

Using the calculated posterior times, a 90% credible interval (lower 5% and upper 95%) was calculated and plotted ontop if the temperature data.



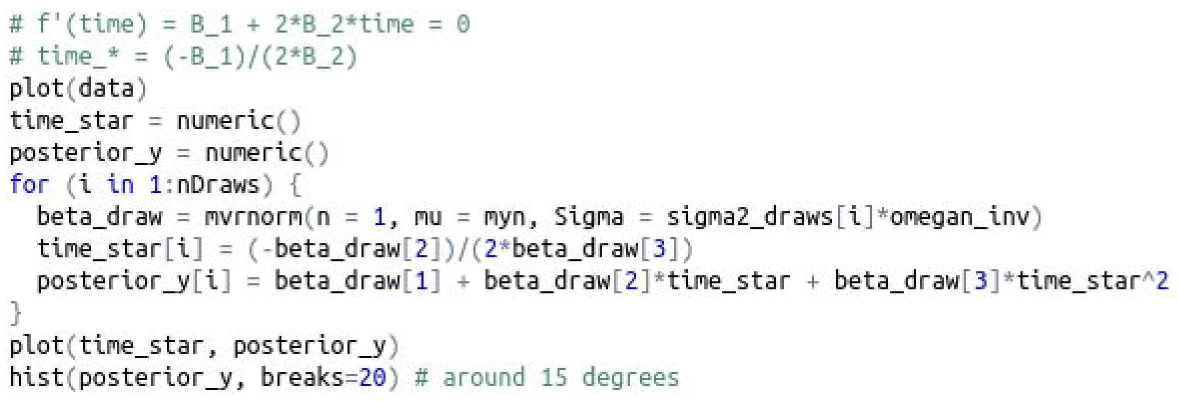
```
posterior_CI = apply(X = posterior_y, MARGIN=2, FUN=function(x) quantile(x,c(0.05, 0.95), na.rm=T))
lines(data$time, posterior_CI[2,], col="green")
lines(data$time, posterior_CI[1,], col="green")
```



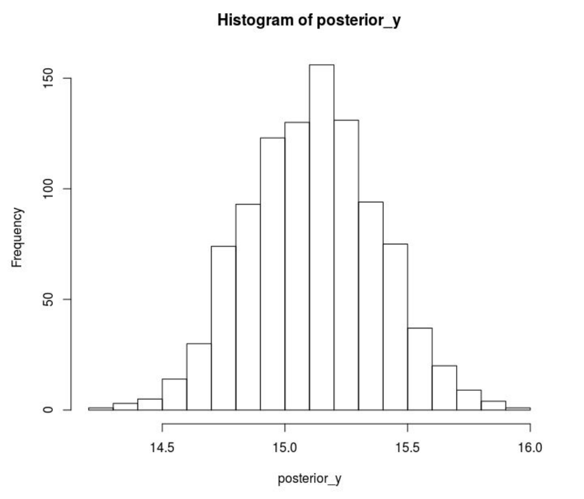
Looking at the interval above, it's seems like very few of the data points is contained by the interval. This is reasonable as the interval is calculated of the regression line and not from data directly.

d)
To locate the time with the highest expected temperature, the first derivative of the quadratic regression function was calculated. By setting it equal to 0, an equation to calculate the time with the maximum temperature was obtained.

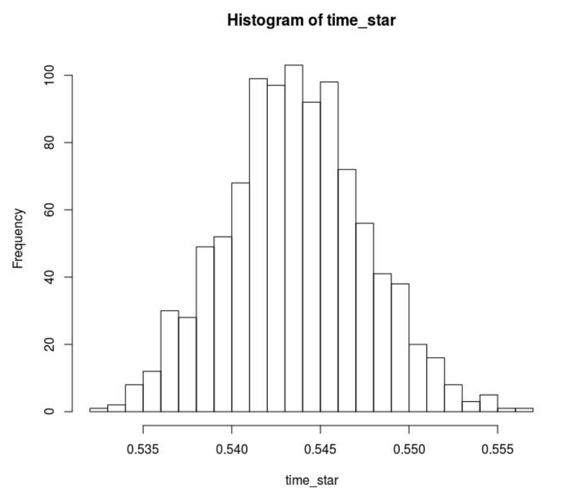
The beta simulations from c) was used to calculate times with the maximum temperatures.



```
# f'(time) = B_1 + 2*B_2*time = 0
# time_* = (-B_1)/(2*B_2)
plot(data)
time_star = numeric()
posterior_y = numeric()
for (i in 1:nDraws) {
  beta_draw = mvrnorm(n = 1, mu = myn, Sigma = sigma2_draws[i]*omegan_inv)
  time_star[i] = (-beta_draw[2])/(2*beta_draw[3])
  posterior_y[i] = beta_draw[1] + beta_draw[2]*time_star + beta_draw[3]*time_star^2
}
plot(time_star, posterior_y)
hist(posterior_y, breaks=20) # around 15 degrees
```



Histogram of the frequency of the maximum temperatures.



Histogram of the frequency of times with maximum temperature

From a visual analysis of the histograms above, it seems like the maximum temperature is happening around time = 0.544.

e)
To mitigate the problem of overfitting when using high order polynomial models, it's reasonable to replace the prior with a ridge regression, where $\mu_o = 0$ and $\Omega_0$ is a diagonal matrix with lambda values. To limit the higher orders, larger values for lambdas corresponding to the higher orders should be chosen:

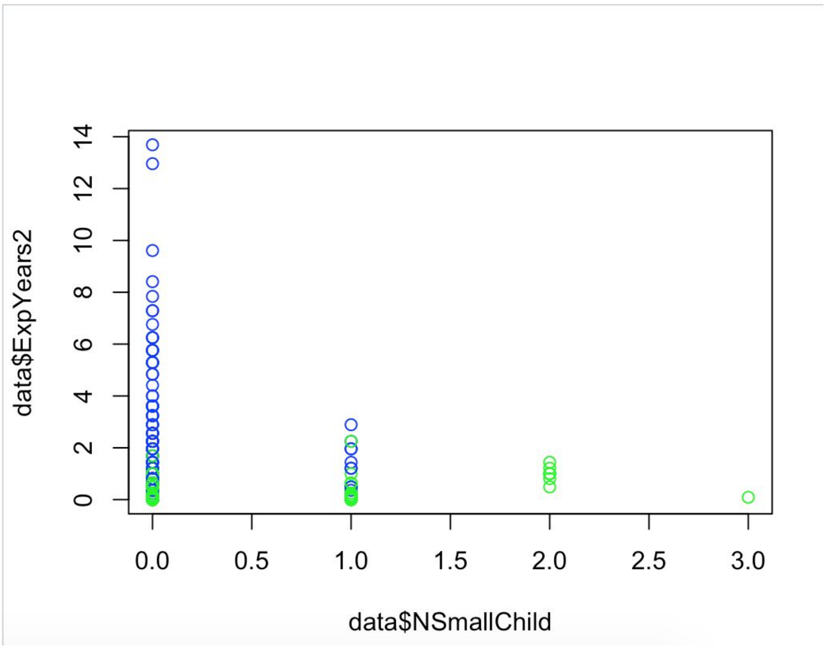
| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | | | | | | |
| | 1 | | | | | |
| | | 2 | | | | |
| | | | 5 | | | |
| | | | | 5 | | |
| | | | | | 10 | |
| | | | | | | 10 |

## 2. Posterior approximation for classification with logistic regression

a) Fitting a regular logistic regression using maximum likelihood estimation with glm:

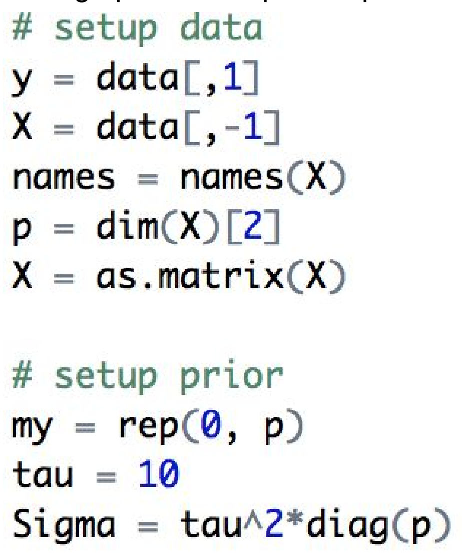
```
# libraries
library("mvtnorm")

# data
data = read.table("WomenWork.dat", header=TRUE)

# a)
# fit model
glm.model <- glm(Work ~ 0 + ., data = data, family = binomial)

# make predictions
pred = glm.model$fitted.values > 0.5

# depict predictions based on # of small children and experience years^2
plot(data$NSmallChild, data$ExpYears2, col=rgb(0,1-pred,pred))
```

Gives the resulting plot:



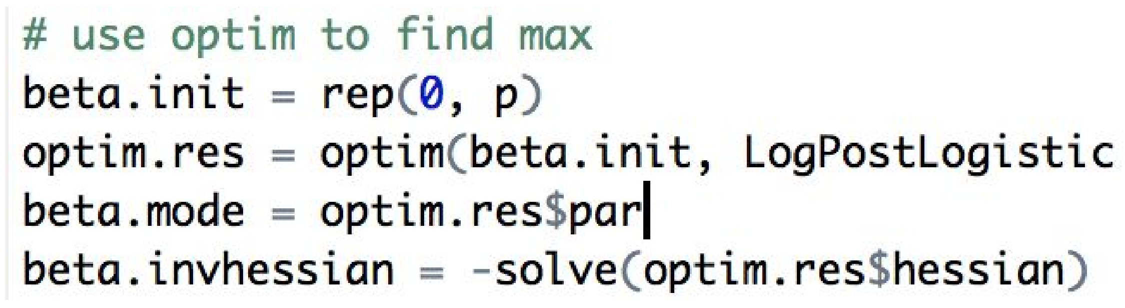
We can see that many small children usually results in a prediction of the woman not working whereas having long work experience results in a prediction of the woman working.

b) Approximating the posterior distribution of the parameters in the logistic regression. Then, estimating the parameters using the mode of the posterior.
Setting up data and prior of parameters:



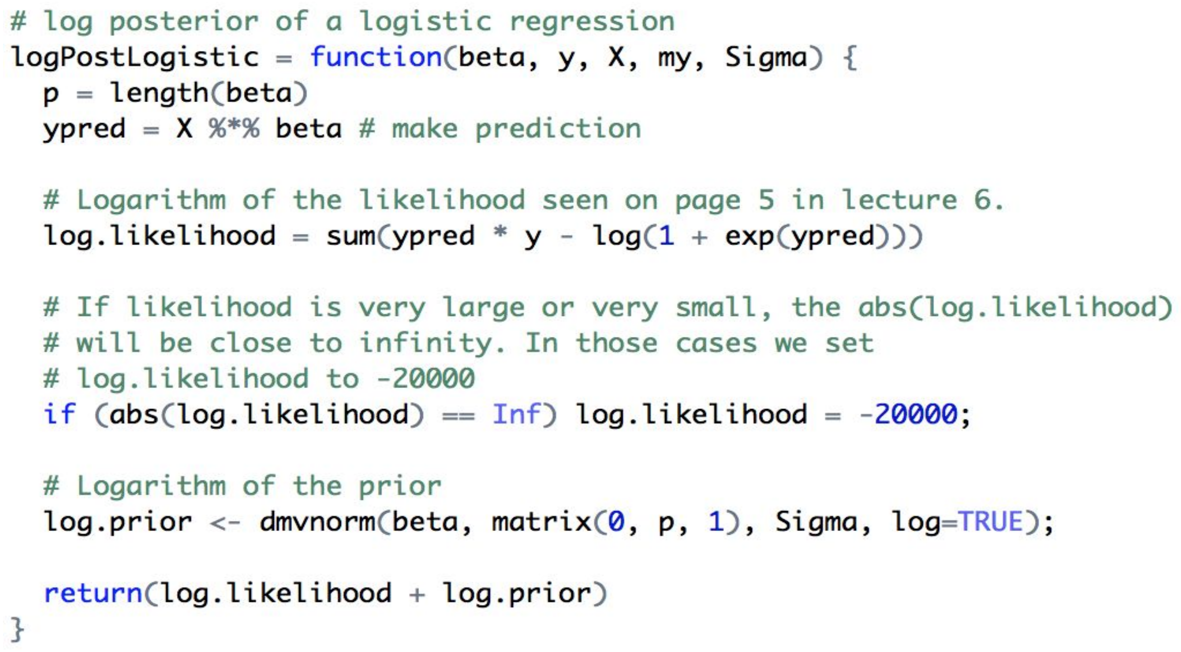
```
# setup data
y = data[,1]
X = data[,-1]
names = names(X)
p = dim(X)[2]
X = as.matrix(X)

# setup prior
my = rep(0, p)
tau = 10
Sigma = tau^2*diag(p)
```

Optimizing the posterior via the log



```
# use optim to find max
beta.init = rep(0, p)
optim.res = optim(beta.init, LogPostLogistic
beta.mode = optim.res$par
beta.invhessian = -solve(optim.res$hessian)
```

Where LogPostLogistic is



```
# log posterior of a logistic regression
logPostLogistic = function(beta, y, X, my, Sigma) {
  p = length(beta)
  ypred = X %*% beta # make prediction

  # Logarithm of the likelihood seen on page 5 in lecture 6.
  log.likelihood = sum(ypred * y - log(1 + exp(ypred)))

  # If likelihood is very large or very small, the abs(log.likelihood)
  # will be close to infinity. In those cases we set
  # log.likelihood to -20000
  if (abs(log.likelihood) == Inf) log.likelihood = -20000;

  # Logarithm of the prior
  log.prior <- dmvnorm(beta, matrix(0, p, 1), Sigma, log=TRUE);

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

Results in Beta mode and :

```
beta.mode = optim.res$par #0.62672884 -0.01979113  0.18021897  0.16756670 -0.14459669 -0.08206561 -1.35913317 -0.02468351
```

```
beta.invhessian = -solve(optim.res$hessian)
#  2.266022568  3.338861e-03 -6.545121e-02 -1.179140e-02  0.0457807243 -3.029345e-02 -0.1887483542 -0.0980239285
#  0.003338861  2.528045e-04 -5.610225e-04 -3.125413e-05  0.0001414915 -3.588562e-05  0.0005066847 -0.0001444223
# -0.065451206 -5.610225e-04  6.218199e-03 -3.558209e-04  0.0018962893 -3.240448e-06 -0.0061345645  0.0017527317
# -0.011791404 -3.125413e-05 -3.558209e-04  4.351716e-03 -0.0142490853 -1.340888e-04 -0.0014689508  0.0005437105
#  0.045780724  1.414915e-04  1.896289e-03 -1.424909e-02  0.0555786706 -3.299398e-04  0.0032082535  0.0005120144
# -0.030293450 -3.588562e-05 -3.240448e-06 -1.340888e-04 -0.0003299398  7.184611e-04  0.0051841611  0.0010952903
# -0.188748354  5.066847e-04 -6.134564e-03 -1.468951e-03  0.0032082535  5.184161e-03  0.1512621814  0.0067688739
# -0.098023929 -1.444223e-04  1.752732e-03  5.437105e-04  0.0005120144  1.095290e-03  0.0067688739  0.0199722657
```

Simulating draws from the posterior to calculate a CI for NSmallChild:

```
# simulate draws
nDraws = 10000
beta.draws = rmvnorm(n = nDraws, mean = beta.mode, sigma = beta.invhessian) # draws from posterior

# CI
beta.NSmallChild.CI = apply(X = beta.draws, MARGIN=2, FUN=function(x) quantile(x,c(0.025, 0.975), na.rm=T))[,7]
```

Results in a 9% CI.NSmallChild= [-2.1315425 -0.6067602] which does not contain 0 and we would then say that this feature is an important determinant of whether a woman works.

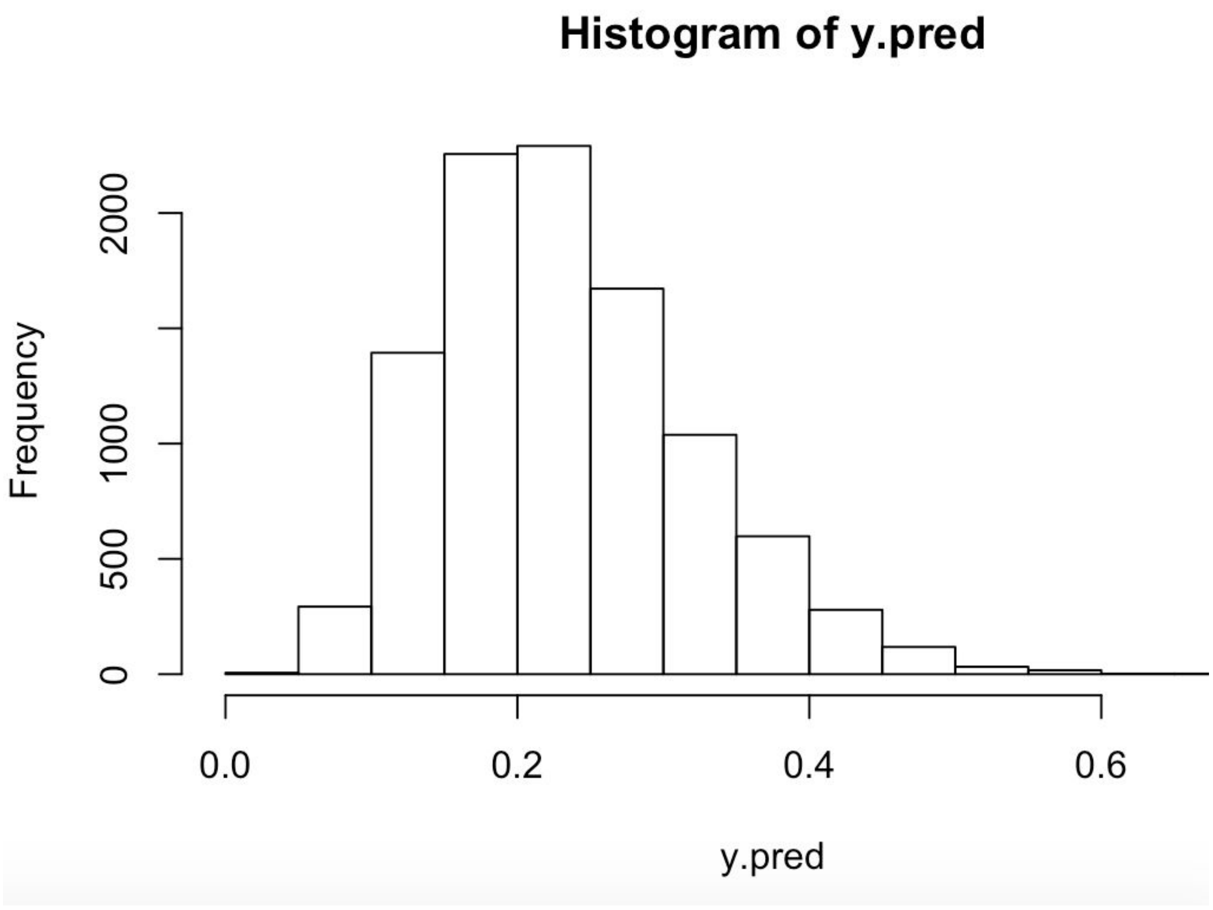
c)
Function simulates from the predictive distribution:

```
logReg = function(woman, mean, Sigma, nDraws) {
  beta.draws = rmvnorm(n = nDraws, mean = mean, sigma = Sigma)
  pred = beta.draws %*% woman
  y.pred = sigmoid(pred)
  return(y.pred)
}
```

Making predictions of a woman:

```r
woman = c(1, 10, 8, 10, (10/10)^2, 40, 1, 1)
y.pred = logReg(woman, beta.mode, beta.invhessian, 10000)
hist(y.pred)
print(mean(ifelse(y.pred>0.5, 1, 0))) #0.0055 ~0.5%
```

Results in histogram:



Where 0.55% of all predictions resulted in prediction of her working, where these are uncertain with a probability slightly over 0.5 as can be seen in the histogram.

# Bayesian Leraning

# Computer lab 3

Elvin Granat (elvgr805)
Max Fischer (maxfi539)

# 1. Normal model, mixture of normal model with semi-conjugate prior.

A dataset consisting of 6919 daily records of precipitation was analyzed using two different models:

a) Normal model
b) Mixture normal model

The recordings range from early 1948 to end of 1983 and includes both rain and snow in units of 1/100 inch. Recordings of zero precipitation are excluded.

**a) Normal model**
The daily precipitation {y1, ..., yn} are assumed to be independent normally distributed:

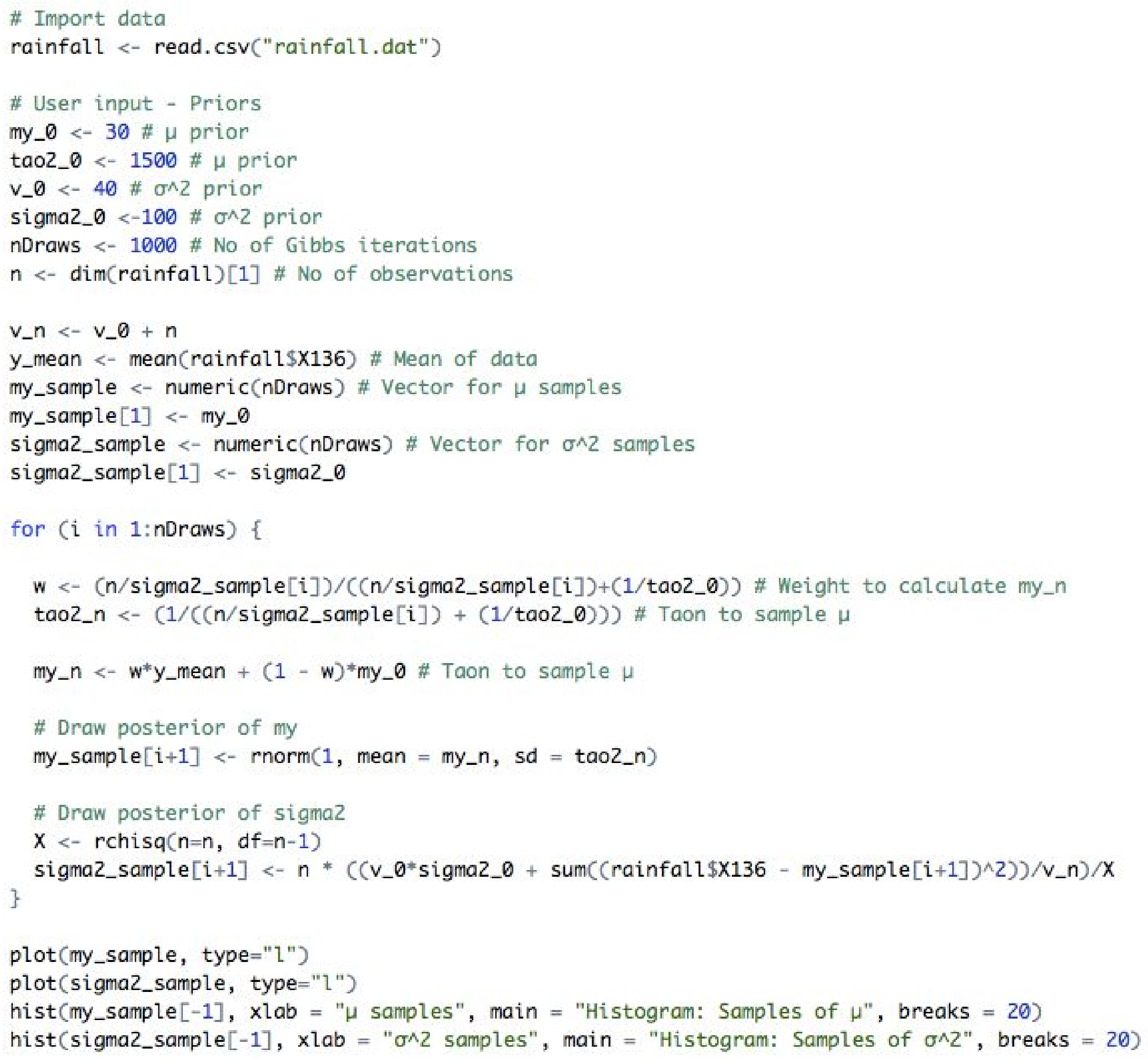
$$y_1, ..., y_n | \mu,\ \sigma^2 \sim N(\mu,\ \sigma^2)$$

$$\mu \sim N(\mu_0, \tau_0^2)$$

$$\sigma^2 \sim Inv - \chi^2(v_0, \sigma_0^2)$$

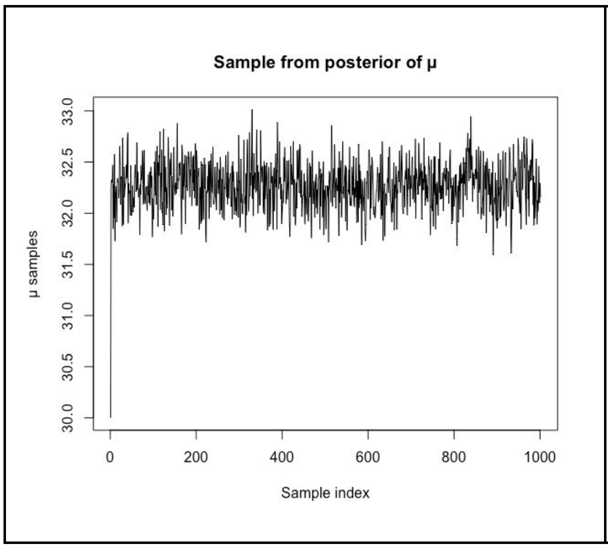
A Gibbs-sampler was implemented that simulated from the joint posterior:

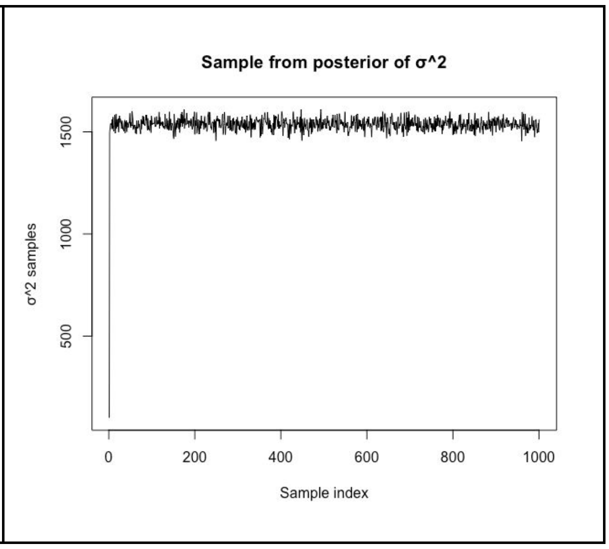
$$p(\mu, \sigma^2 | y_1, ..., y_n)$$

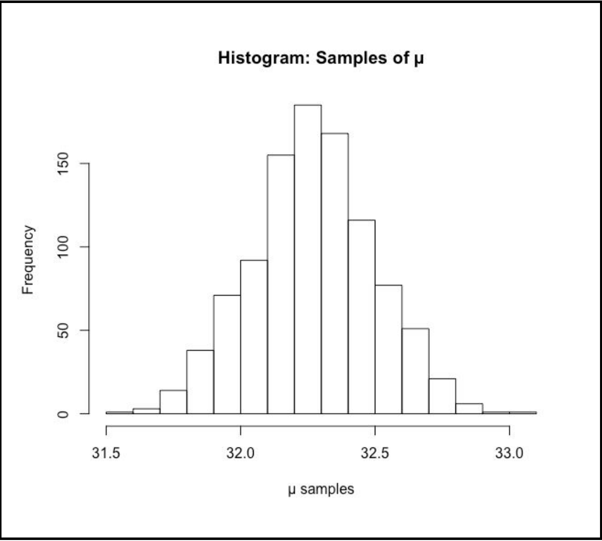


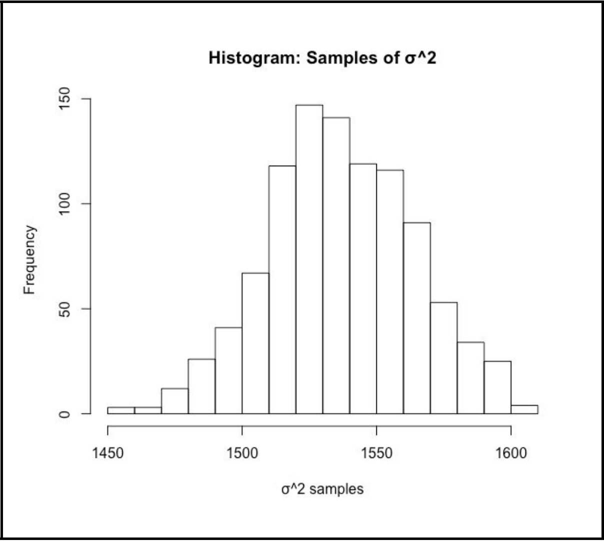
```r
# Import data
rainfall <- read.csv("rainfall.dat")

# User input - Priors
my_0 <- 30 # μ prior
tao2_0 <- 1500 # μ prior
v_0 <- 40 # σ^2 prior
sigma2_0 <-100 # σ^2 prior
nDraws <- 1000 # No of Gibbs iterations
n <- dim(rainfall)[1] # No of observations

v_n <- v_0 + n
y_mean <- mean(rainfall$X136) # Mean of data
my_sample <- numeric(nDraws) # Vector for μ samples
my_sample[1] <- my_0
sigma2_sample <- numeric(nDraws) # Vector for σ^2 samples
sigma2_sample[1] <- sigma2_0

for (i in 1:nDraws) {

  w <- (n/sigma2_sample[i])/((n/sigma2_sample[i])+(1/tao2_0)) # Weight to calculate my_n
  tao2_n <- (1/((n/sigma2_sample[i]) + (1/tao2_0))) # Taon to sample μ

  my_n <- w*y_mean + (1 - w)*my_0 # Taon to sample μ

  # Draw posterior of my
  my_sample[i+1] <- rnorm(1, mean = my_n, sd = tao2_n)

  # Draw posterior of sigma2
  X <- rchisq(n=n, df=n-1)
  sigma2_sample[i+1] <- n * ((v_0*sigma2_0 + sum((rainfall$X136 - my_sample[i+1])^2))/v_n)/X
}

plot(my_sample, type="l")
plot(sigma2_sample, type="l")
hist(my_sample[-1], xlab = "μ samples", main = "Histogram: Samples of μ", breaks = 20)
hist(sigma2_sample[-1], xlab = "σ^2 samples", main = "Histogram: Samples of σ^2", breaks = 20)
```

```
for (i in 1:10000) {
  ## mu sampling
  # mun
  w = (n / sigma2.sample) / ( n / sigma2.sample + 1 / tau0^2 )
  mun = w * mean(x) + (1 - w) / mu0
  # taun2
  taun2 = 1 / (n / sigma2.sample + 1 / tau0^2)
  # sample mu
  mu.sample = rnorm(1, mean = mun, sd = sqrt(taun2))
  #mu.sample = rexp(1, rate = 1 / mun) if you want to try exponential
  mu.samples[i] = mu.sample

  ## sigma sampling
  # vn
  vn = n + v0
  # sigma2n
  sigma2n = (v0 * sigma0^2 + sum((x - mu.sample)^2)) / (n + v0)
  chi2 = rchisq(vn, n=1)
  sigma2.sample = vn * sigma2n / chi2
  sigma2.samples[i] = sigma2.sample
}
# plot of the trajectories
plot(mu.samples, type="l")
plot(sigma2.samples, type="l")
```









According to the trajectory plots above, it seems like the Gibbs samplers of $\mu$ and $\sigma^2$ converges quite quickly towards:

$$\mu \approx 32.25$$

$$\sigma^2 \approx 1525$$

This is also confirmed by the histogram plots.

**b) Mixture normal model**
The daily precipitation {y1, ..., yn} are now assumed to follow an iid two-component mixture of normal models:

$$p(y_1, ..., y_n | \mu,\ \sigma^2, \pi)\ =\ \pi * N(y_1, ..., y_n | \mu_1,\ \sigma_1^2) + (1-\pi) * N(y_1, ..., y_n | \mu_2,\ \sigma_2^2)$$

$$\mu = (\mu_1, \mu_2)$$
$$\sigma^2 = (\sigma_1^2, \sigma_2^2)$$

A Gibbs sampling data augmentation algorithm was provided to analyze the daily precipitation data.

The following priors was chosen:

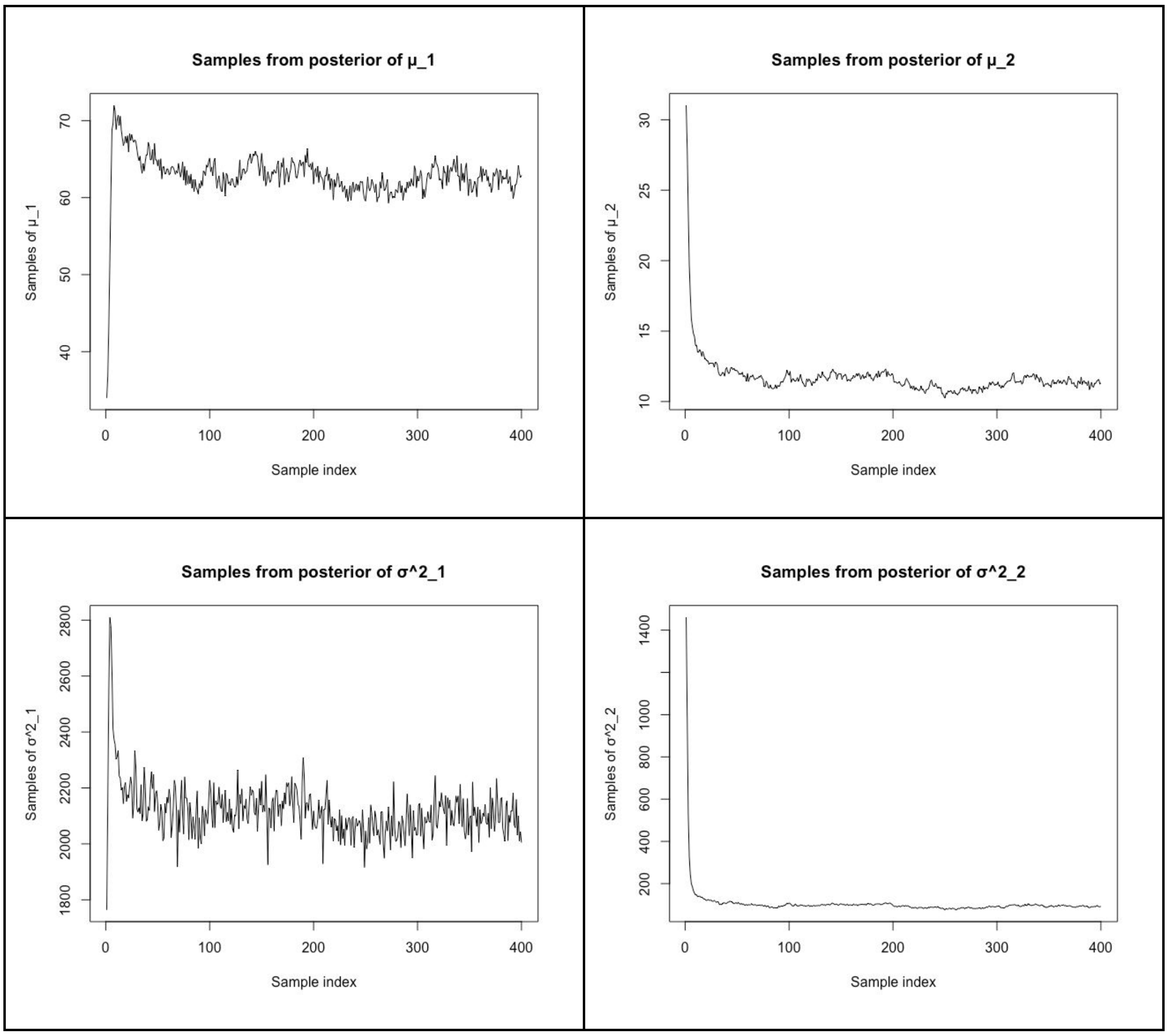
$$\alpha = (\alpha_1, \alpha_2)\ =\ (10,\ 10)$$
$$\mu_0 = (\mu_{0,1}, \mu_{0,2})\ =\ (mean(data),\ mean(data))\ =\ (32.2681,\ 32.2681)$$
$$\tau_0^2 = (\tau_{0,1}^2, \tau_{0,2}^2)\ =\ (100,\ 100)$$
$$\sigma_0^2 = (\sigma_{0,1}^2, \sigma_{0,2}^2)\ =\ (40^2,\ 40^2) = (1600,\ 1600)$$
$$\nu_0 = (\nu_{0,1},\ \nu_{0,2})\ =\ (5,\ 5)$$

Initially, $(\pi_1, \pi_2)$ oscillated up to around (0.70, 0.30) at iteration no 8, allocating the observations as following:

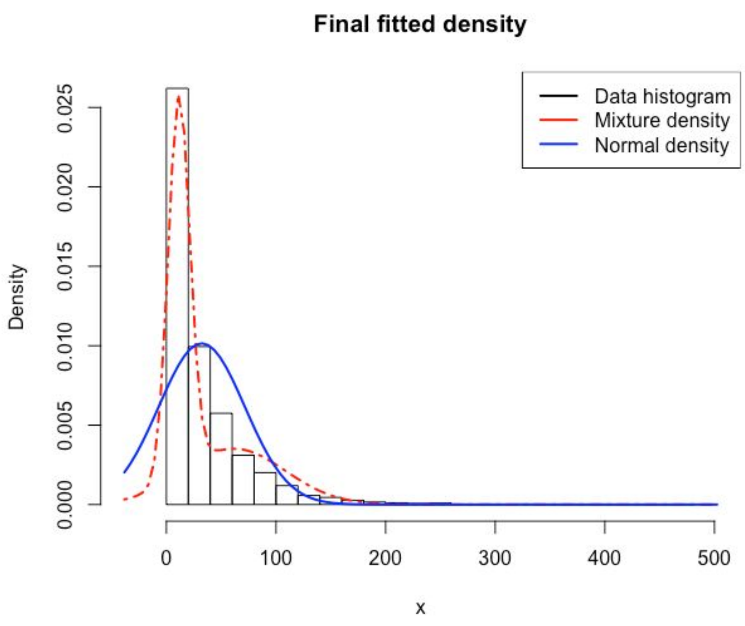
(2155, 4764)

After that, $(\pi_1, \pi_2)$ slowly converged towards numbers close to (0.60, 0.30), which happened around iteration no 30, allocating the observations as following:

(2723, 4196)



## c) Graphical comparison



## 2. Time series models in Stan

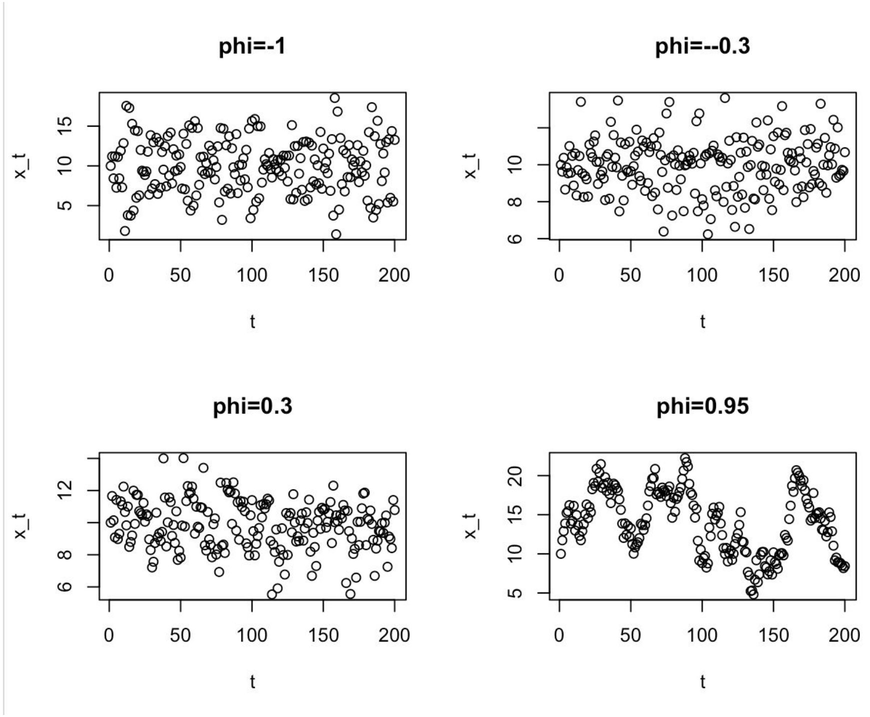
a) Function that simulates from an AR(1) process with given values for μ, σ and values of Φ between -1 and 1 (this is where an AR(1) process is stable)

$$x_t = \mu + \phi\left(x_{t-1} - \mu\right) + \varepsilon_t, \quad \varepsilon_t \overset{iid}{\sim} N\left(0, \sigma^2\right)$$

Simulating T values, starting from t = 1 and continuing until t = T = 200.

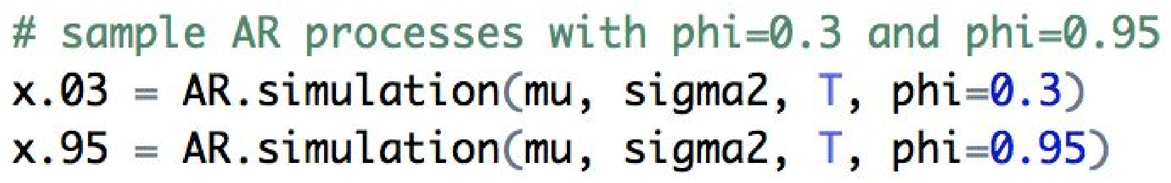
```
AR.simulation = function(mu, sigma2, T, phi) {
  x.sample = numeric()
  x.sample[1] = mu
  for(i in 2:T) {
    x.sample[i] = mu + phi * (x.sample[i-1] - mu) + rnorm(1, mean = 0, sd = sqrt(sigma2))
  }
  return(x.sample)
}

### Implementation
# a)

phis = seq(-1,1,0.05)
n = length(phis)
x.samples = matrix(nrow=n, ncol=T, 0)
for(i in 1:n) {
  x.samples[i,] = AR.simulation(mu, sigma2, T, phis[i])
}
```

Plotting some results:



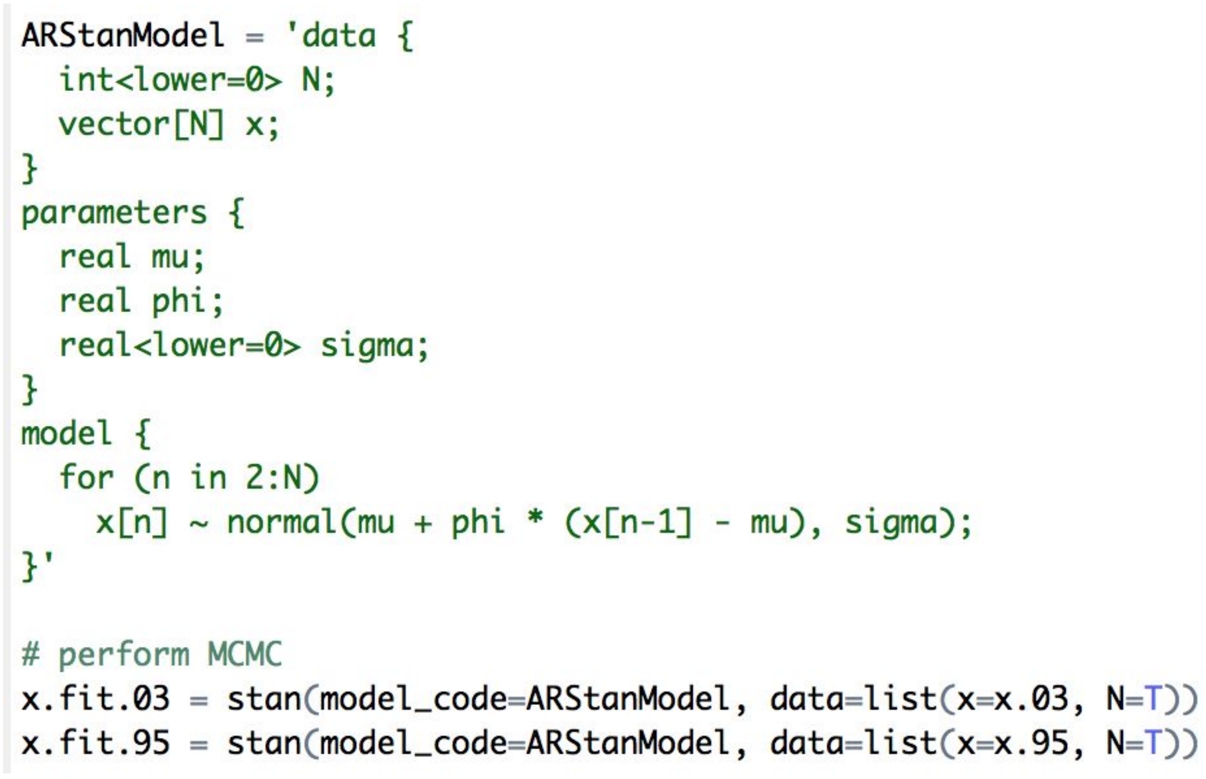
Negative Φ resutls in the x_t values alternating a more around the mean value whereas for positive, especially larger, Φ values the curve follows a temporary trend.

b) Now μ, σ and Φ are treated as unknown. Given data sampled using the function above with same values as before with Φ=0.3 and Φ=0.95.



```
# sample AR processes with phi=0.3 and phi=0.95
x.03 = AR.simulation(mu, sigma2, T, phi=0.3)
x.95 = AR.simulation(mu, sigma2, T, phi=0.95)
```

Estimating the values from  μ, σ and Φ using MCMC in Rstan:



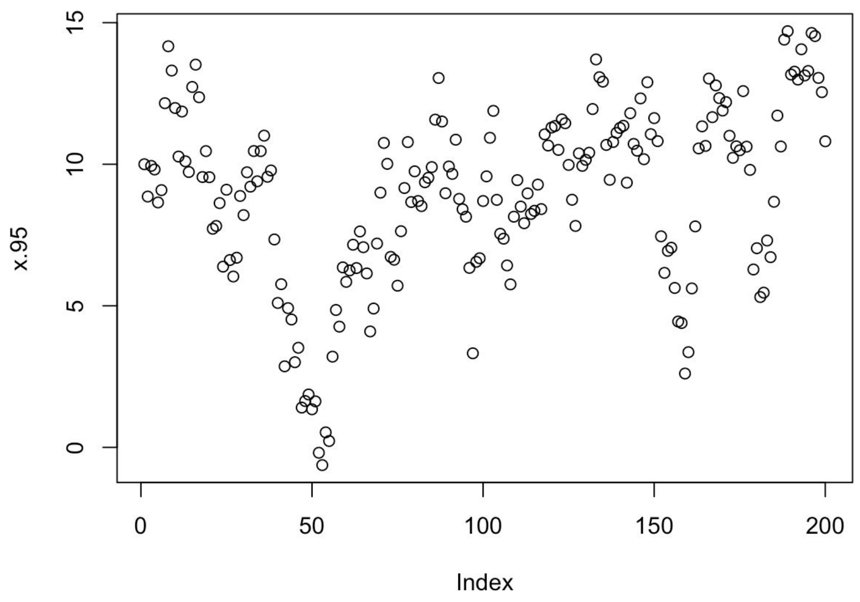
```
ARStanModel = 'data {
  int<lower=0> N;
  vector[N] x;
}
parameters {
  real mu;
  real phi;
  real<lower=0> sigma;
}
model {
  for (n in 2:N)
    x[n] ~ normal(mu + phi * (x[n-1] - mu), sigma);
}'

# perform MCMC
x.fit.03 = stan(model_code=ARStanModel, data=list(x=x.03, N=T))
x.fit.95 = stan(model_code=ARStanModel, data=list(x=x.95, N=T))
```

The number of effective samples, posterior mean and credible interval for each parameter.

| Φ = 0.3 | #effective samples | Posterior mean | CI |
|---|---|---|---|
| μ | 2929 | 10.123 | (9.79, 10.45) |
| σ | 3335 | 1.438 | (1.31, 1.58) |
| Φ | 3151 | 0.371 | (0.241, 0.510) |

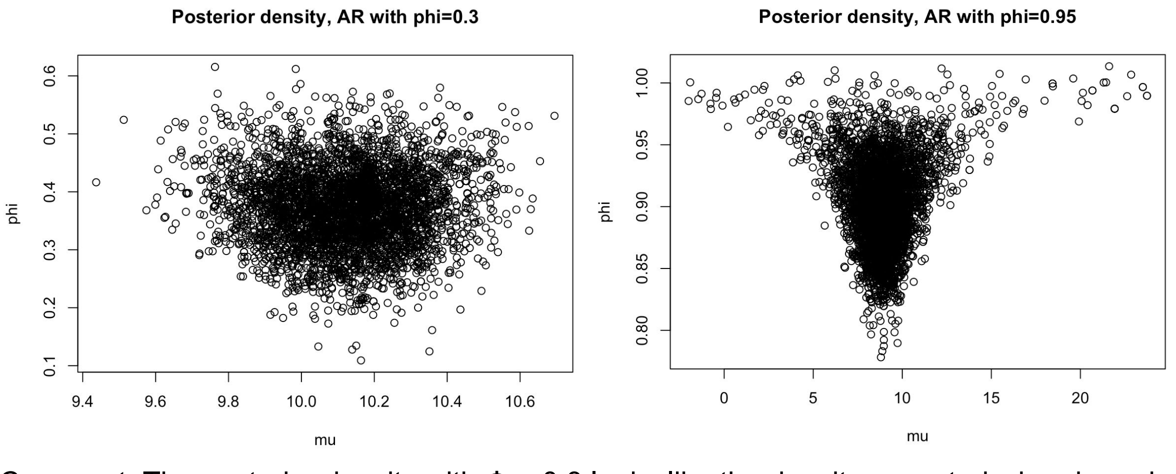
| Φ = 0.95 | #effective samples | Mean | CI |
|---|---|---|---|
| μ | 594 | 8.960 | (5.85, 12.34) |
| σ | 2887 | 1.479 | (1.34, 1.64) |
| Φ | 1517 | 0.903 | (0.838, 0.979) |

Both attempts estimate the parameter values well. However, the number of effective samples are lower and the credible intervals are wider for μ with the dataset based on Φ = 0.95. We believe this is because the trendy pattern that can easily to far below and above the μ value may result in a skewed dataset given only 200 observations in the sample.



Plotting the sampled data shows that a estimation of μ below 10 is not unreasonable given how many data points are below 10 given the argument above.

Plotting the joint posterior of Φ and μ:



Comment: The posterior density with Φ = 0.3 looks like the density espected when based on a normal distribution. Φ = 0.95 however looks to be a bit skewed in its density for Φ values close to 1. This may be because:

- the given dataset gives reasonable probabilities when calculating α (acceptance probabiliy) with Φ values  close to 1
- that a sampled Φ close to 1 results in high likelihood regardless of sampled μ value.

This is also expressed in the #effective samples for μ and Φ compared to Φ = 0.3 seen in the table above, where both of these parameters have a somewhat low #effective samples, especially μ with 594 out of 4000 effective samples. This calls for either a better proposed distribution when sampling or more samples to witness the "true" shape of the posterior.

c) Model the number of campylobacter infections in four week intervals:

$$c_t | x_t \sim Poisson\left(\exp\left(x_t\right)\right)$$

Where c_t is contioned on a latent AR(1) process (= we do not have any observations from the AR process). This results in that the AR process x_t should be seen as a parameter.

Estimating the model using Rstan:

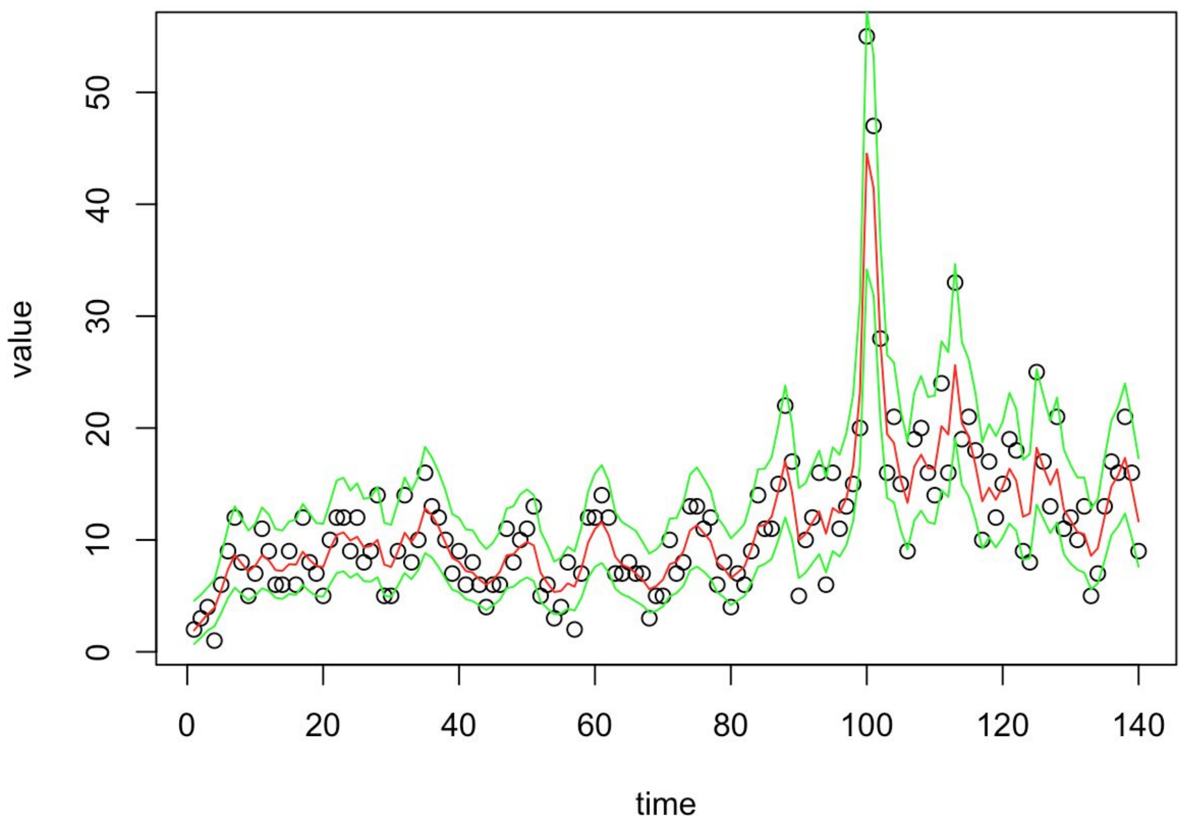
```
data.campy = read.table("campy.dat", header=TRUE)[,1]
N = length(data.campy)

PoissonARStanModel = '
data {
  int<lower=0> N;
  int c[N];
}
parameters {
  real mu;
  real phi;
  real<lower=0> sigma;
  vector[N] x;
}
model {
  // mu ~ normal();
  // sigma2 ~ scaled_inv_chi_square();
  // phi ~ normal(0, 0.6);
  phi ~ uniform(-1,1);
  for (n in 2:N)
    x[n] ~ normal(mu + phi * (x[n-1] - mu), sigma);
  for (n in 1:N)
    c[n] ~ poisson(exp(x[n]));
}'
c.fit = stan(model_code=PoissonARStanModel, data=list(N = N, c = data.campy))
```

We use uninformative priors for sigma and mu since we do not know how the latent variable behaves what so ever. For Φ however we tried using a uniform prior given that we know it should be between -1 and 1, we also tried a normal prior with mean 0 and a reasonable variance, the results were very similar though.

Confidence interval for the latent intensity $\theta_t = \exp(x_t)$:

```
X.summary = summary(c.fit)$summary[-c(1,2,3),]

# extract CI and mean of x
x.upper = X.summary[,"97.5%"]
x.lower = X.summary[,"2.5%"]
x.mean = X.summary[,"mean"]

# plot data, mean and CI of the latent intensity
plot(data.campy, xlab="time", ylab="value")
lines(exp(x.upper), col="green")
lines(exp(x.lower), col="green")
lines(exp(x.mean), col="red")
```

Confidence interval with the data points:

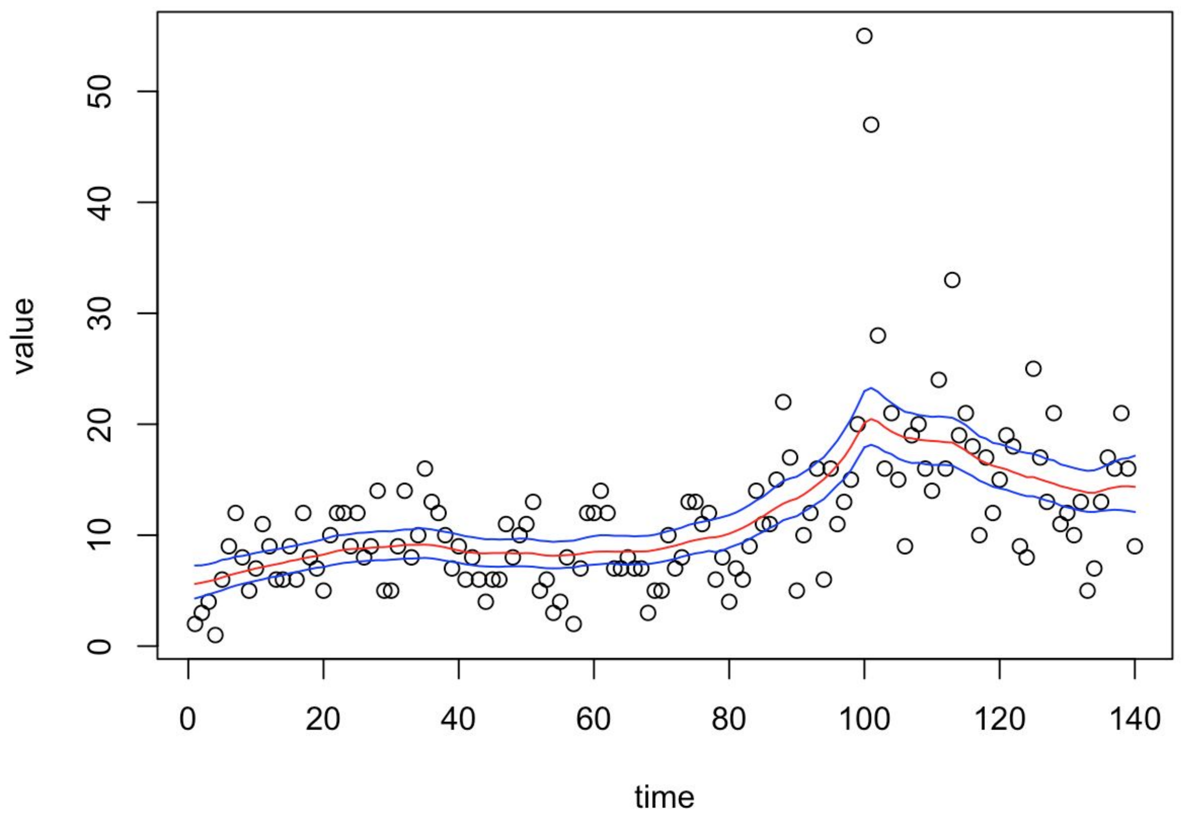


The latent intensity follows the data which is nice, however it seems a bit overfitted given that it follows the data too much. It is very volatile, a smoothness factor seems reasonable.

d) Given the shape of the confidence intervals we added a prior to the variance $\sigma^2$ telling the sampler that the value of $\sigma^2$ should not be too large. We used a Scaled-Inv $\chi^2$ estimating the $\sigma^2$ to be closer to 0:

```
model {
  // mu ~ normal();
  // phi ~ normal(0, 0.6);
  sigma2 ~ scaled_inv_chi_square(N, 0.03);
  phi ~ uniform(-1,1);
  for (n in 2:N)
    x[n] ~ normal(mu + phi * (x[n-1] - mu), sqrt(sigma2));
  for (n in 1:N)
    c[n] ~ poisson(exp(x[n]));
}'
```

Resulting plot of the confidence interval of the latent intensity:



These curves are a lot smoother while still following the pattern of the data, hence it seems like a more reasonable model of the data since it is not as overfitted in case of future predictions.

# 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 $\beta$ 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:

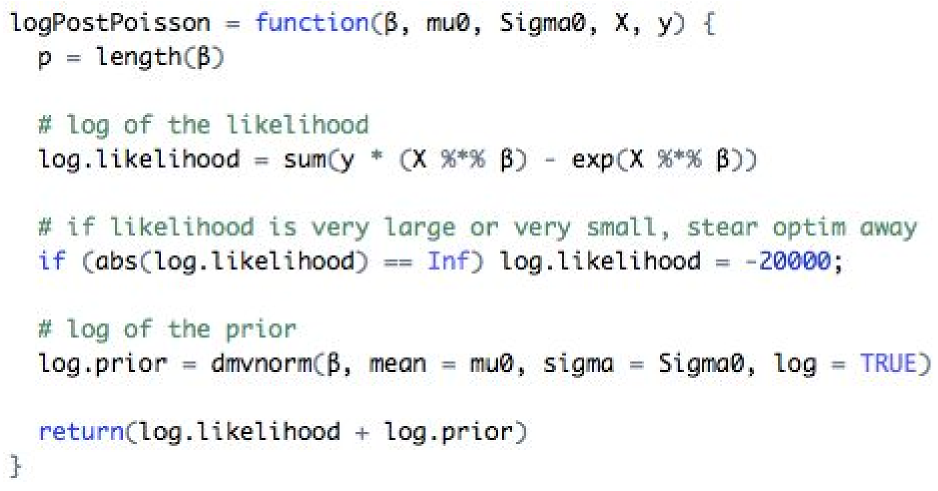
- (Intercept) (99,9%)
- VerifyID (99,9%)
- Sealed (99,9%)
- LogBook (99,9%)
- MinBidShare (99,9%)
- MajBlem (95%)

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 $\beta$ , $J_y^{-1}(\widehat{\beta})$ : Inverse negative Hessian at the mode of $\beta$ .

To calculate the mode of $\beta$ and the inverse negative Hessian at the mode of $\beta$ 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:**

| | $\beta_1$ | $\beta_2$ | $\beta_3$ | $\beta_4$ | $\beta_5$ | $\beta_6$ | $\beta_7$ | $\beta_8$ | $\beta_9$ |
|---|---|---|---|---|---|---|---|---|---|
| **Mode** | 1.069841 | -0.020512 | -0.393006 | 0.443556 | -0.052466 | -0.221238 | 0.070697 | -0.120218 | -1.891985 |

**Inverse negative Hessian:**

| | $\beta_1$ | $\beta_2$ | $\beta_3$ | $\beta_4$ | $\beta_5$ | $\beta_6$ | $\beta_7$ | $\beta_8$ | $\beta_9$ |
|---|---|---|---|---|---|---|---|---|---|
| $\beta_1$ | 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 |
| $\beta_2$ | -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 |
| $\beta_3$ | -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 |
| $\beta_{41}$ | -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 |
| $\beta_5$ | -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 |
| $\beta_6$ | -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 |
| $\beta_7$ | -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 |
| $\beta_8$ | 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 |
| $\beta_9$ | 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))

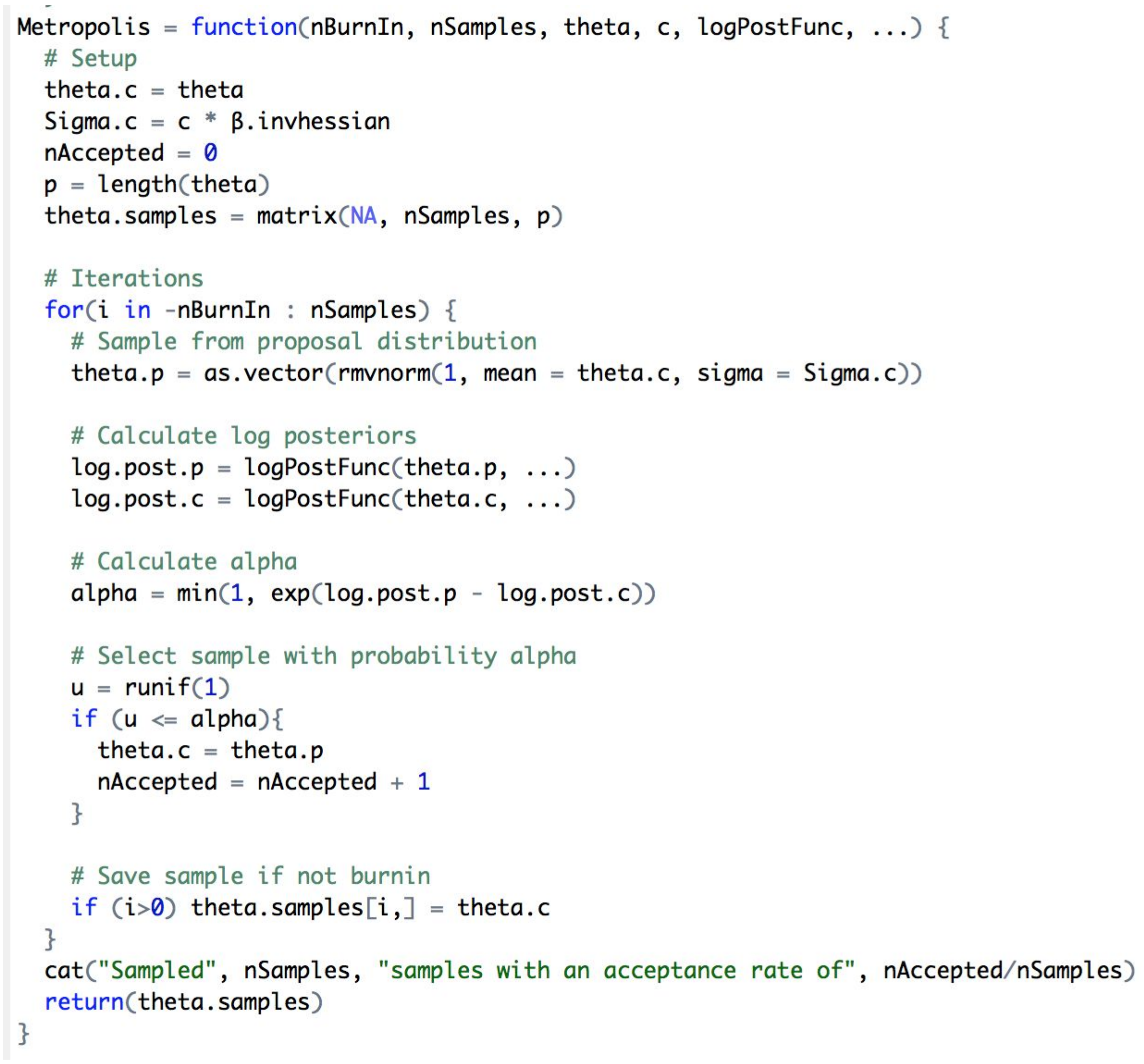
And c is a given tuning parameter.

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

The Metropolis function takes in a number of parameters:

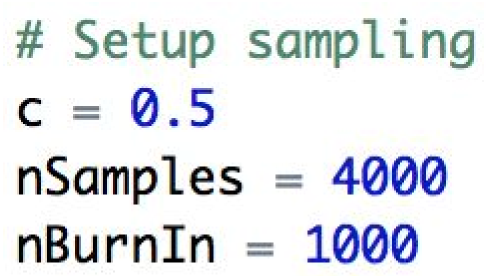
**nBurnIn** = number of burnin iterations of the algorithm
**nSamples** = number of samples after the burnin iterations
**theta** = the parameter of which to sample, used to evaluate the posterior density
**c** = tuning parameter for the proposal density
**logPostFunc** = computes the log posterior density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

Final Metropolis function is as follows:



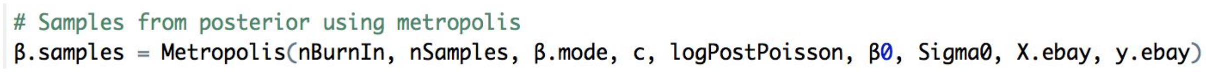
```
Metropolis = function(nBurnIn, nSamples, theta, c, logPostFunc, ...) {
  # Setup
  theta.c = theta
  Sigma.c = c * β.invhessian
  nAccepted = 0
  p = length(theta)
  theta.samples = matrix(NA, nSamples, p)

  # 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)
```

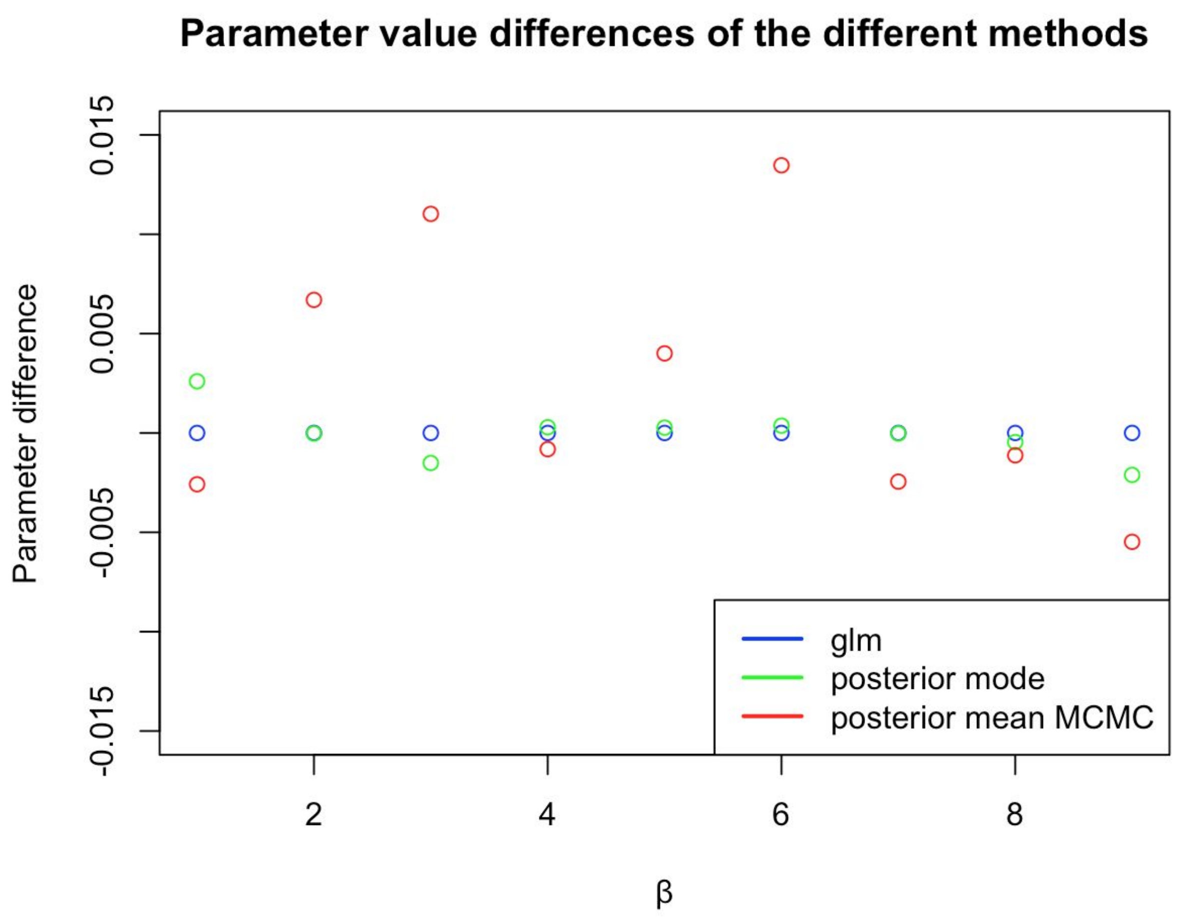
Results:

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

**Mean of beta:**

| | $\beta_1$ | $\beta_2$ | $\beta_3$ | $\beta_4$ | $\beta_5$ | $\beta_6$ | $\beta_7$ | $\beta_8$ | $\beta_9$ |
|---|---|---|---|---|---|---|---|---|---|
| **Mean** | 1.075024 | -0.027237 | -0.405541 | 0.444659 | -0.056203 | -0.234342 | 0.073121 | -0.119552 | -1.888617 |

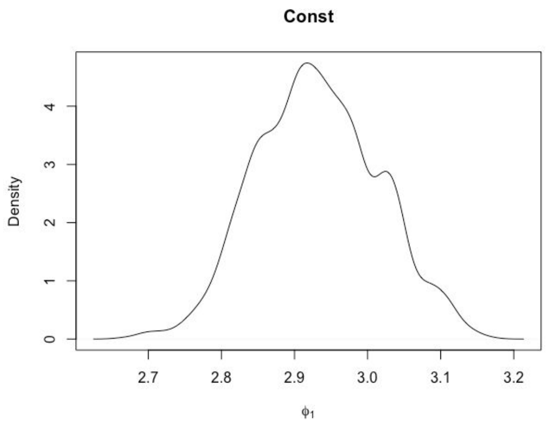
To graphically compare the parameter values of the different estimation methods we plot the value difference of the parameters. With the result of glm method as the middle values. Results in the following plot:

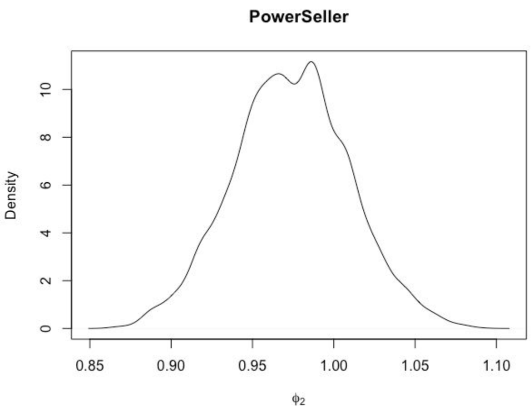


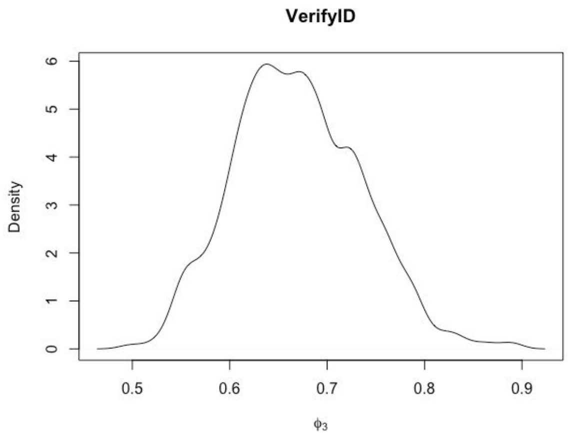
We can see that glm and posterior mode using optim reults in similar results whereas MCMC gives very different parameter values for some paramaters.

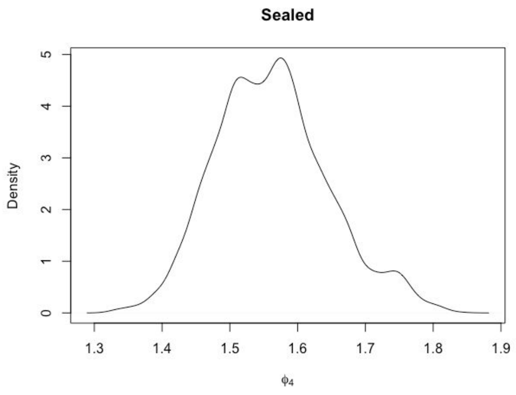
To visualize the convergence MCMC we compute the posterior distribution of:

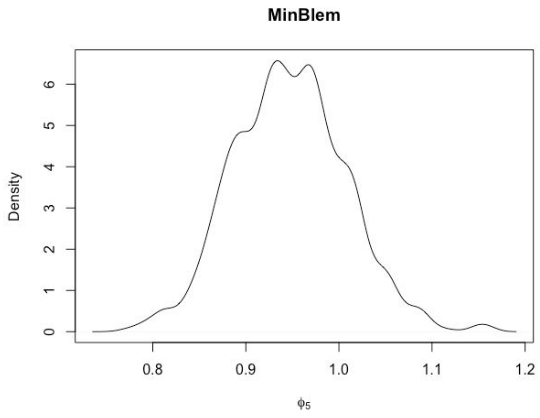
$\phi_j = \exp(\beta_j)$ for the sampled betas.

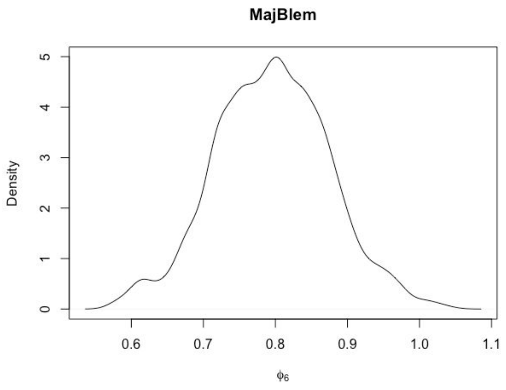


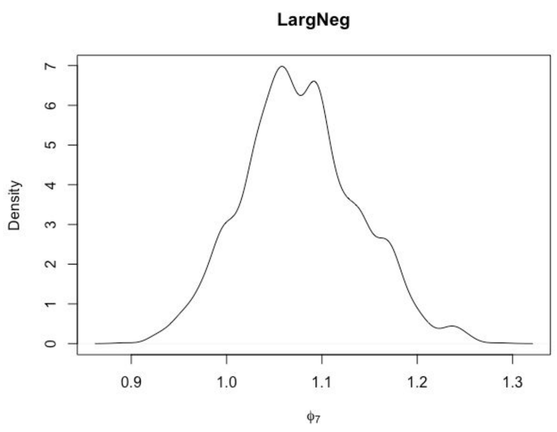


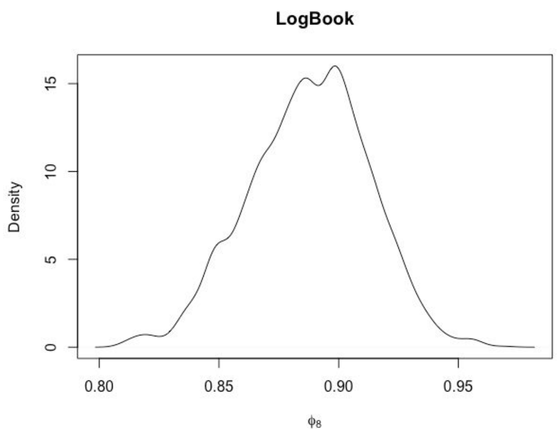


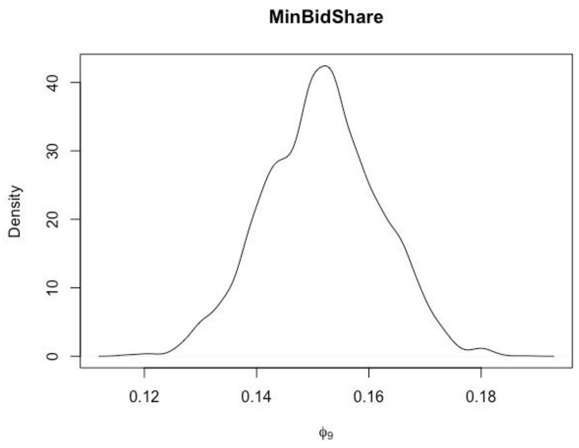












Which looks like reasonable posterior distributions. Interesting that we can see that the range of the parameter values that were deemed significant in a) don't cross 1 with any of the samples. Whereas the not so significant covariates' parameters have most of its mass close to 1.

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

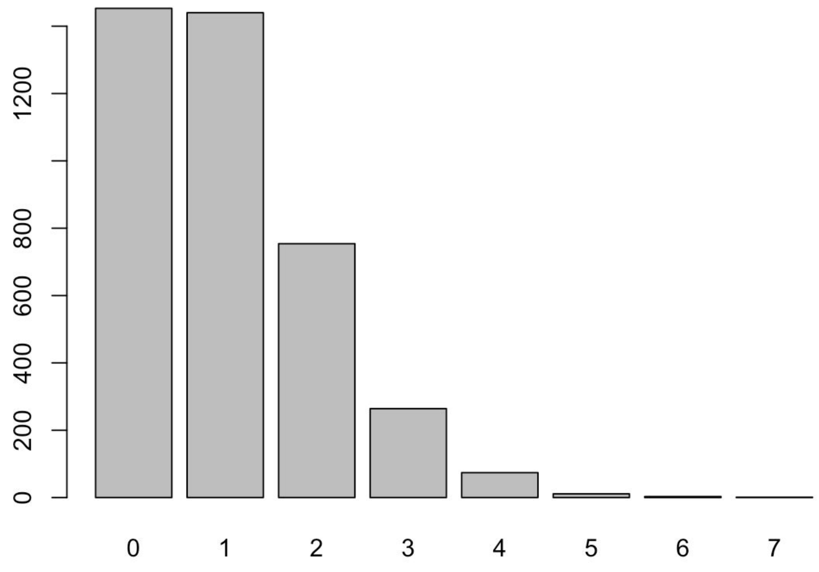
Given auction:

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

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.