Lab1 Report

Ludvig Noring, Michael Sörsäter

April 11, 2017

1. Bernoulli

(a)

Using the sample with 14 successes and 6 failures and a Beta prior with $\alpha = \beta = 2$ we get the posterior $\theta|y \sim Beta(16, 8)$ from which we draw random numbers. As the number of draws increases the sampled mean and standard deviation converges towards the theoretical mean and standard deviation for the posterior. This can be seen in Figure 1 and Figure 2.

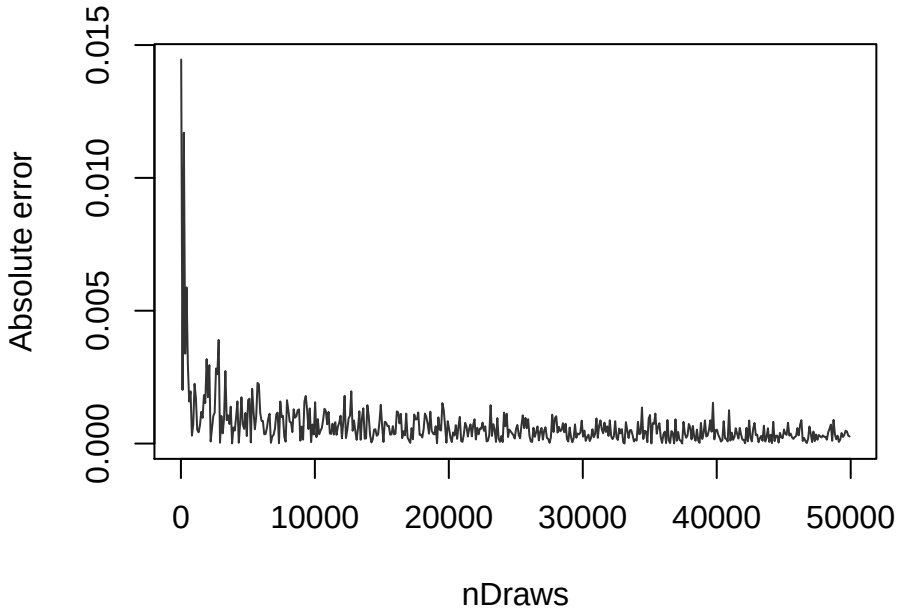

Figure 1: Absolute error of the sampled and theoretical mean

(b)

With 10 000 simulated draws we compute the posterior probability to $Pr(\theta < 0.4|y) = 0.35\%$. This can be compared with the theoretical value 0.397%.

With just so few as 10 000 draws we get a probability close to the exact value.

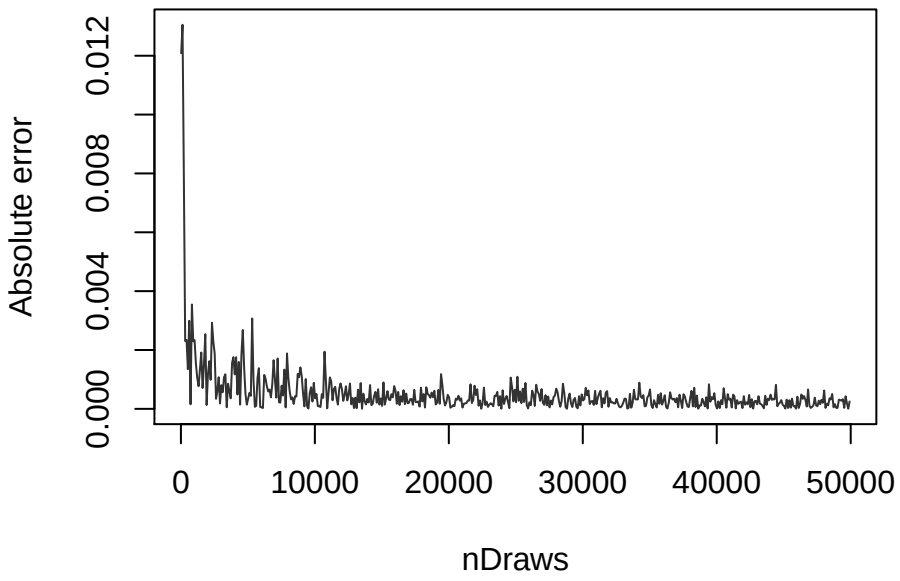

Figure 2: Absolute error of the sampled and theoretical standard deviation

(c)

When using simulation it is easy to compute the log-odds of the posterior distribution. The plot is seen in Figure 3..

2. Log-normal distribution and the Gini coefficient

(a)

10 000 draws are simulated from the posterior $Inv - \chi^2(n, \tau^2)$. The result is plotted with the theoretical distribution and is seen in Figure 4. They are very similar.

(b)

We calculate the Gini coefficient for different values of σ and plot the result which can be seen in Figure 5.

(c)

The equal tails interval and highest posterior density interval are computed from the draws in 2b. The density have a very long right tail which influence the equal tail interval. Therefore the highest posterior density seems to fit the data better. The plot with the credible intervals can be seen in Figure 6.

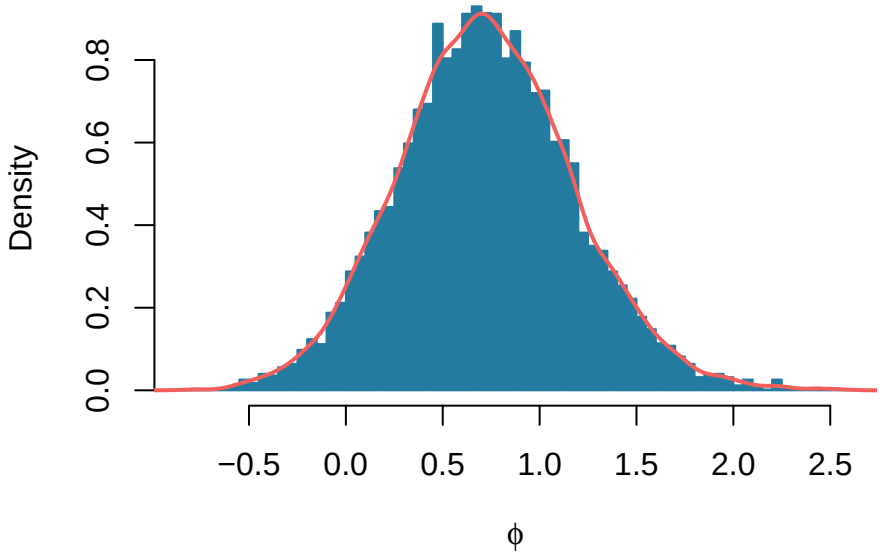

Figure 3: Log-odds

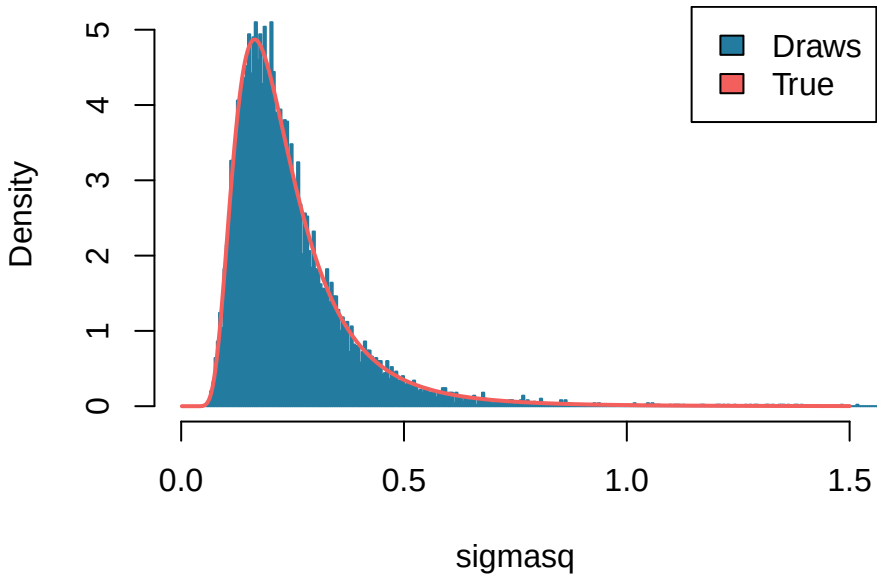

Figure 4: $Inv - \chi^2(n, \tau^2)$ Simulated and theoretical

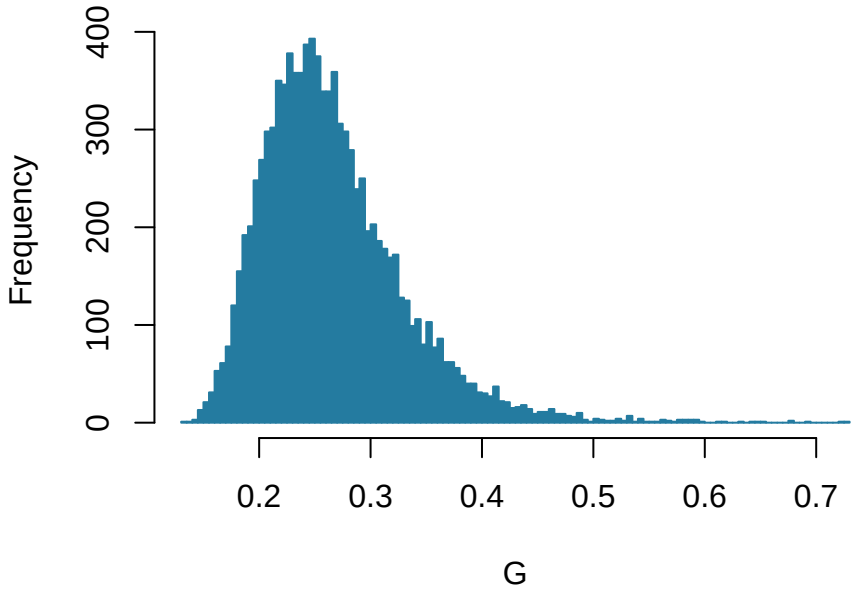

Figure 5: Gini coefficient

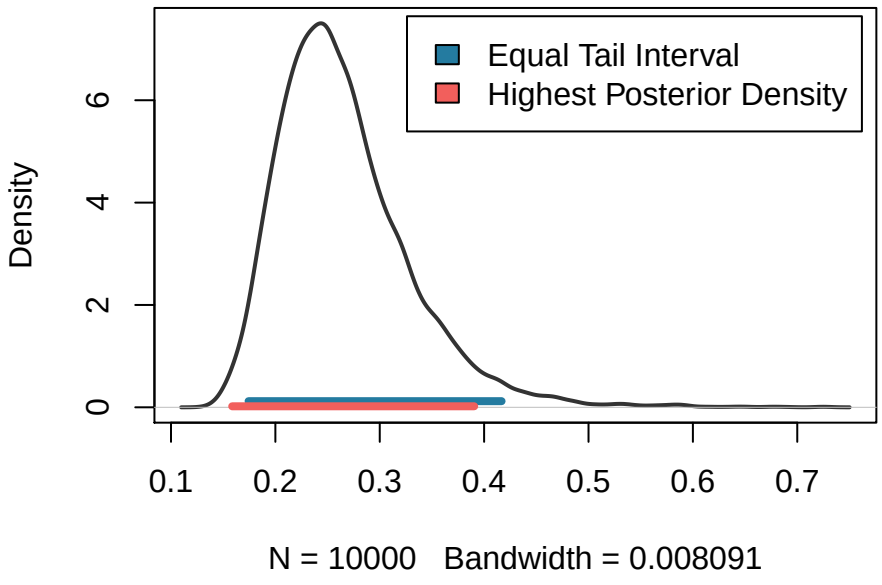

Figure 6: Credible intervals

3. Von Mises distribution

The posterior distribution is calculated by multiplying the individual observations together with the prior for a fine grid of κ values. The mode is calculated for the posterior. The plot can be found in Figure 7.

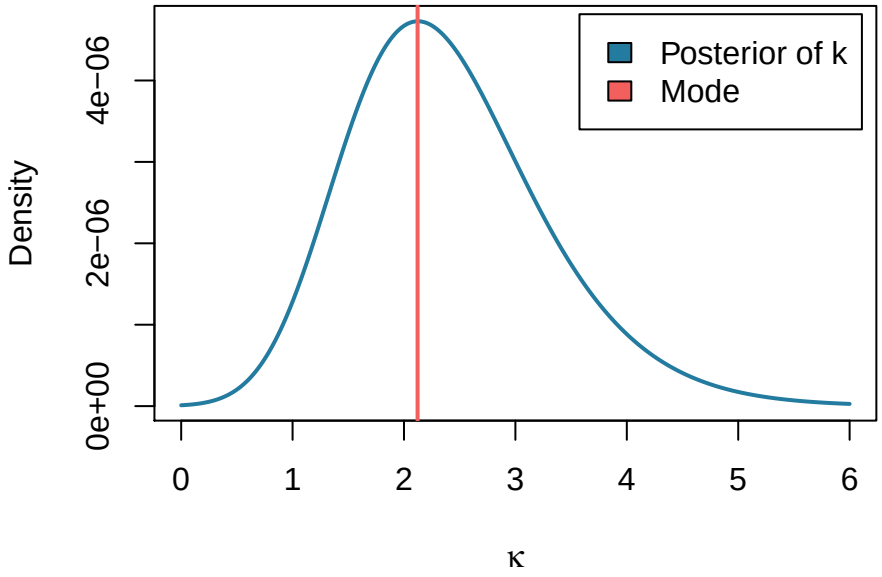

Figure 7: Von Mises distribution

Appendix A - Code for assignment 1

```
set.seed(12345)
col1 = '#247ba0'
col2 = '#f25f5c'
col3 = '#333333'

imgw = 5
imgh = 4

a = 2
b = 2
s = 14
n = 20

xGrid <- seq(0.001, 0.999, 0.001)

## 1a
draw = function(nDraws){
  sample = rbeta(nDraws, a+s, b+(n-s))
  mean_sample = mean(sample)
  sd_sample = sd(sample)

  return (c(mean_sample, sd_sample))
}

alpha = a + s
beta = b + (n-s)

intervals = seq(20,50000,100)
data = sapply(intervals, draw)

true = dbeta(xGrid, a+s, b+(n-s))
mean_true = alpha / (alpha + beta)
sd_true = sqrt( (alpha*beta) / ((alpha+beta)^2*(alpha+beta+1)) )

# Mean
pdf("plots/1-converge_mean.pdf", width=imgw, height=imgh)
  plot(intervals, abs(data[1,]-mean_true), type='l', col=col3,
    xlab = 'nDraws', ylab='Absolute error')
dev.off()

# SD
pdf("plots/1-converge_sd.pdf", width=imgw, height=imgh)
  plot(intervals, abs(data[2,]-sd_true), type='l', col=col3,
    xlab = 'nDraws', ylab='Absolute error')
dev.off()

## 1b
nDraws = 10000
posterior = rbeta(nDraws, a+s, b+(n-s))
true = pbeta(0.4, a+s, b+(n-s))
```

```r
# Simulated value
message((sum(posterior <= 0.4) / length(posterior)) * 100)
# Theoretical value
message(round(true*100, 3))

## 1c
nDraws = 10000
posterior = rbeta(nDraws, a+s, b+(n-s))
phi = log(posterior / (1 - posterior))

pdf("plots/1-phi.pdf", width=imgw, height=imgh)
  hist(phi, 100, prob=TRUE, col=col1, border=col1, main='', xlab=expression(phi))
  lines(density(phi), lwd=2, col=col2)
dev.off()
```

Appendix B - Code for assignment 2

```
col1 = '#247ba0'
col2 = '#f25f5c'
col3 = '#333333'
imgw = 5
imgh = 4

mu = 3.5
obs = c(14, 25, 45, 25, 30, 33, 19, 50, 34, 67)
n = length(obs)

tausq = sum((log(obs) - mu)^2) / (n)

chiDraw = function(nDraws){
  #draws = rchisq(nDraws, n)
  #sigmasq = n * tausq / draws
  sigmasq = rinvchisq(nDraws, n, tausq)
  return (sigmasq)
}

nDraws = 10000
sigmasq = chiDraw(nDraws)

s = seq(0, 1.5, length.out = 1000)
#exp_part = exp( ((-n*tausq) / (2*s)) ) / (s^(1+n/2))
#constant = ((tausq * n/2)^(n/2)) / (factorial(n/2-1))
#                                    gamma(n/2)
#PDF = constant * exp_part
PDF = dinvchisq(s, n, tausq)

pdf('plots/2-chi-squared.pdf', width=imgw, height=imgh)
  hist(sigmasq, 500, col=col1, border=col1, main='', freq=FALSE, xlim=c(0,1.5))
  lines(s, CDF, col=col2, lwd=2)
  legend('topright', c('Draws', 'True'), fill=c(col1, col2))
dev.off()

## 2.b Gini
vals = sqrt(sigmasq/2)
G = 2 * pnorm(vals) - 1

pdf('plots/2-g.pdf', width=imgw, height=imgh)
  hist(G, 100, col=col1, border=col1, main='')
dev.off()


## 2.c Credible Intervals

# Equal tail interval
x_eti = quantile(G, probs=c(0.025, 0.975))

# Highest Posterior Density
dg = density(G)
```

```r
# Sort data on y decending values
ordered_x = dg$x[order(-dg$y)]
ordered_y = dg$y[order(-dg$y)]

cur = 0
for(i in 1:length(dg$y)){
  cur = cur + ordered_y[i]
  if(cur / sum(dg$y) >= 0.95){
    break
  }
}
x_hpd = c(min(ordered_x[1:i]), max(ordered_x[1:i]))

pdf('plots/2-credible-intervals.pdf', width=imgw, height=imgh)
  plot(dg, col=col3, lwd=2, main="")
  lines(x_eti, rep(0.12, 2), col=col1, lwd=4)
  lines(x_hpd, rep(0.02, 2), col=col2, lwd=4)
  #points(ordered_x[1:i], rep(0, i), cex=0.1)
  legend("topright",
         legend = c("Equal Tail Interval","Highest Posterior Density"),
         fill = c(col1, col2),
         inset = 0.02)
dev.off()
```

Appendix C - Code for assignment 3

```
col1 = '#247ba0'
col2 = '#f25f5c'
imgw = 5
imgh = 4

y = c(-2.44, 2.14, 2.54, 1.83, 2.02, 2.33, -2.79, 2.23, 2.07, 2.02)
mu = 2.39

von_misen = function(y, kappa, mu){
  I_zero = besselI(x=kappa, nu=0)
  numerator = exp(kappa * cos(y - mu))
  return (numerator / (2 * pi * I_zero))
}

exponential = function(kappa, lambda=1){
  return (lambda * exp(-lambda*kappa))
}

calculate_posterior = function(y, kappa, mu) {
  prior = exponential(kappa)
  likelihood = prod(von_misen(y, kappa, mu))
  return (prior * likelihood)
}

# To calculate the mode
get_mode = function(vec, kappas) {
  vec = round(vec, 10)
  uniqv = unique(vec)
  mode = uniqv[which.max(tabulate(match(vec, uniqv)))]

  mode_idx = which(vec == mode)[1]
  mode_x = kappas[mode_idx]

  return (mode_x)
}

kappas = seq(0, 6, 0.001)

posterior = sapply(kappas, calculate_posterior, y=y, mu=mu)
mode = get_mode(posterior, kappas)

pdf("plots/3-posterior-distribution.pdf", width=imgw, height=imgh)
  plot(kappas, posterior, type='l', lwd=2, col=col1,
       xlab=expression(kappa), ylab="Density")
  abline(v=mode, col=col2, lwd=2)
  legend('topright', c('Posterior of k', 'Mode'), fill=c(col1, col2), inset=0.02)
dev.off()
```

Lab2 Report

Ludvig Noring, Michael Sörsäter

April 26, 2017

1. Linear and polynomial regression

a & b

We tried to reason about the hyperparameters. After simulating draws from the model we updated the hyperparameters and ended up with the following values. The resulting regression curves can be seen in Figure 1.

$$\mu_0 = (-10, 100, -100)$$

$$\Omega_0 = I$$

$$v_0 = 366 - 3 = 363$$

$$\sigma_0^2 = 5$$

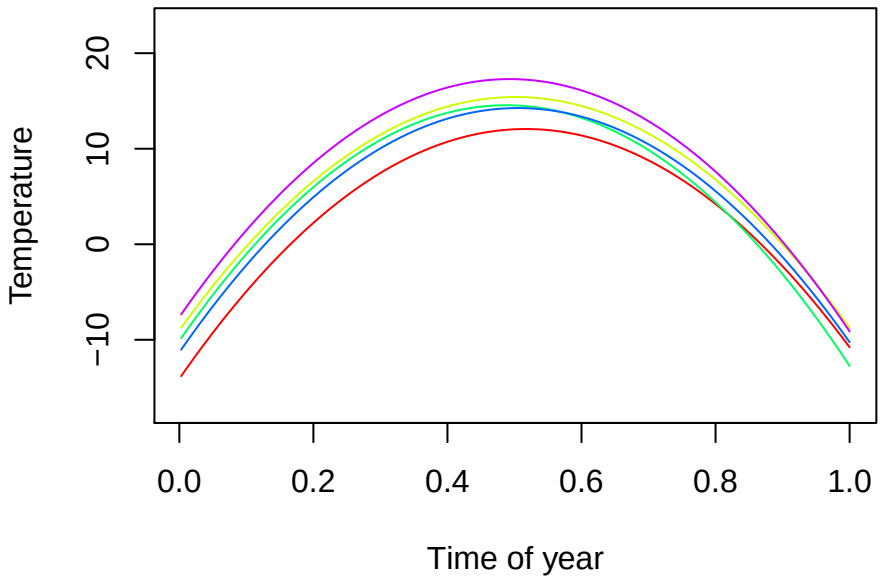

Figure 1: Regression curves

The temperature starts at around $-10°C$ and reaches its peak in the middle of the year at $20°C$ and finally ends at the same starting value $-10°C$.

c

We simulated draws from the joint posterior and calculated the posterior mean of the β values. The plot of the posterior mean together with the credible intervals can be seen in Figure 2.

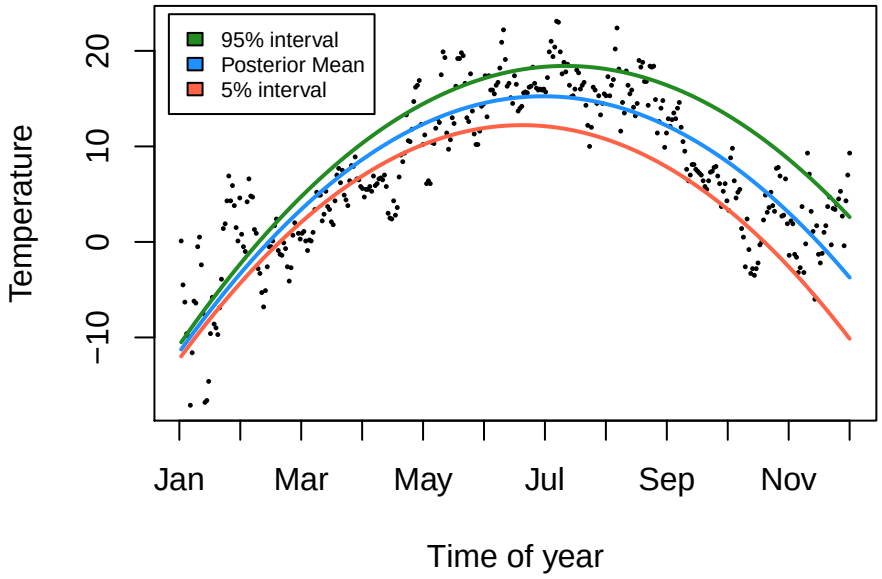

Figure 2: Scatter plot with credible intervals

d

We calculated the maximum temperature with the following formula. This resulted in the numeric value 0.5429473 which corresponds to the date 17th of July.

$$\tilde{x} = -\frac{\beta_1}{2\beta_2}$$

e

When estimating with a polynomial with degree 7 we supress polynomial by setting the coefficients of the higher orders to zero. Below is μ_0 and Ω_0.

	1	2	3	4	5	6	7	8
1	-10.00	100.00	-100.00	0.00	0.00	0.00	0.00	0.00

Table 1: Mu

	1	2	3	4	5	6	7	8
1	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
3	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 2: Omega

2. Posterior approximation for classification with logistic regression

a

Using the glm function to fit the logistic regression results in the coefficients seen in table 3.

	Constant	HusbandInc	EducYears	ExpYears	ExpYears2	Age	NSmallChild	NBigChild
1	0.64	-0.02	0.18	0.17	-0.14	-0.08	-1.36	-0.03

Table 3: Coefficents using glm

b

We used the function optim to find optimal values for $\tilde{\beta}$ and $J_y^{-1}(\tilde{\beta})$. The result can be show in the tables below. For the feature NSmallChild the 95% credible interval is shown in Figure 3.

	1	2	3	4	5	6	7	8
1	0.63	-0.02	0.18	0.17	-0.14	-0.08	-1.36	-0.02

Table 4: Beta tilde

c

From the optim ä'aal β values we simulated 1000 draws and for each of them calculated the probability to be used in the Bernoulli draws. From the Bernoulli draws we plotted the probability that the woman is working. The plot can be seen in Figure 4.

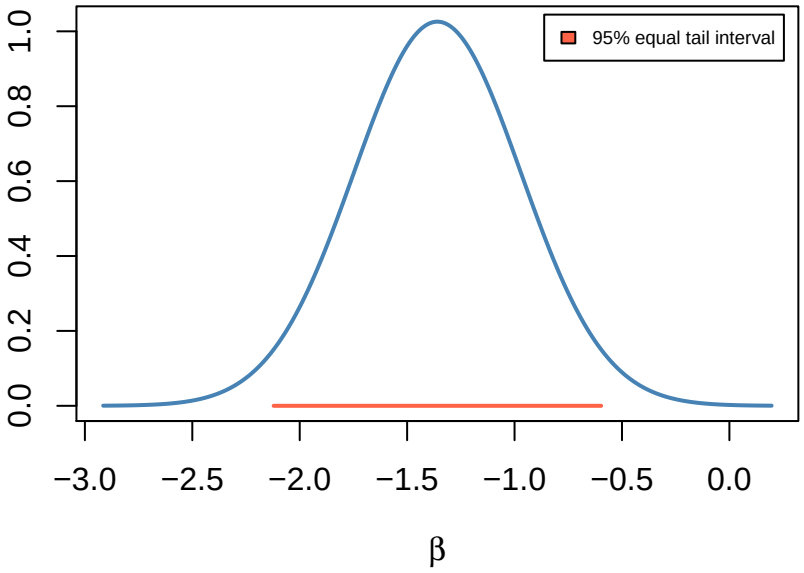

Figure 3: Equal Tail intervals for NSmallChild

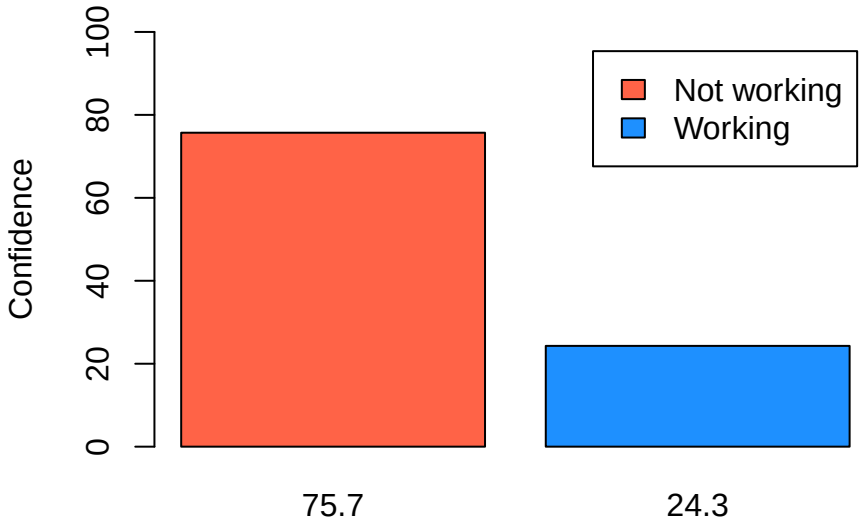

Figure 4: Working confidence

	1	2	3	4	5	6	7	8
1	2.27	0.00	-0.07	-0.01	0.05	-0.03	-0.19	-0.10
2	0.00	0.00	-0.00	-0.00	0.00	-0.00	0.00	-0.00
3	-0.07	-0.00	0.01	-0.00	0.00	-0.00	-0.01	0.00
4	-0.01	-0.00	-0.00	0.00	-0.01	-0.00	-0.00	0.00
5	0.05	0.00	0.00	-0.01	0.06	-0.00	0.00	0.00
6	-0.03	-0.00	-0.00	-0.00	-0.00	0.00	0.01	0.00
7	-0.19	0.00	-0.01	-0.00	0.00	0.01	0.15	0.01
8	-0.10	-0.00	0.00	0.00	0.00	0.00	0.01	0.02

Table 5: Observed hessian evaluated at the posterior mode

Appendix A - Code for assignment 1

```
require('polynom')
require('MASS')
require('geoR')
set.seed(1234567890)
imgw = 5
imgh = 4

data = read.table('TempLinkoping2016.txt', header=TRUE)
n = nrow(data)

# a
quad = lm(temp ~ time + I(time^2), data=data)
coefs = polynomial(c(quad$coefficients))

#plot(data$time, data$temp, pch=20)
#lines(data$time, predict(coefs, data$time))

# Hyper parameters
mu_0 = c(-10, 100, -100)
omega_0 = diag(3) # 3 = nParas
v_0 = n - 3 # 3 = nParas
sigma2_0 = 5

# b
cols = rainbow(5)

pdf('plots/hyper.pdf', width=imgw, height=imgh)
  plot(data, pch=20, cex=0.3, type='n', xlab='Time of year', ylab='Temperature')
  for (i in 1:5){
    sigma2 = rinvchisq(1, df=v_0, scale=sigma2_0)
    betas = mvrnorm(n = 1, mu = mu_0, Sigma=sigma2 * solve(omega_0))
    coefs = polynomial(betas)
    lines(data$time, predict(coefs, data$time), col=cols[i])
  }
dev.off()

# c
y = data$temp
X = as.matrix(data.frame(beta0=rep(1, n), beta1=data$time, beta2=data$time^2))

beta_hat = solve(t(X) %*% X) %*% t(X) %*% y
mu_n = solve(t(X) %*% X + omega_0) %*% (t(X) %*% X %*% beta_hat + omega_0 %*% mu_0)
omega_n = t(X) %*% X + omega_0
v_n = v_0 + n
v_n.sigma2_n = v_0 * sigma2_0 +
                (t(y) %*% y +
                t(mu_0) %*% omega_0 %*% mu_0 -
                t(mu_n) %*% omega_n %*% mu_n)

sigma2 = rinvchisq(1000, df=v_n, scale=v_n.sigma2_n/v_n)
betas = sapply(sigma2, function(sigma2){mvrnorm(n=1, mu=mu_n, Sigma=sigma2*solve(omega_n))})
```

```
betas_mean = apply(betas, 1, mean)
print(betas_mean)

ci = apply(betas, 1, quantile, probs=c(0.05, 0.95))

cols = c('forestgreen', 'dodgerblue', 'tomato')
pdf('plots/posterior.pdf', width=imgw, height=imgh)
  plot(data, pch=20, cex=0.3, xlab='Time of year', ylab='Temperature', xaxt='n')
  lines(data$time, predict(polynomial(ci[2,]), data$time), col=cols[1], lwd=2)
  lines(data$time, predict(polynomial(betas_mean), data$time), col=cols[2], lwd=2)
  lines(data$time, predict(polynomial(ci[1,]), data$time), col=cols[3], lwd=2)
  legend('topleft', inset=0.02,
         legend = c('95% interval', 'Posterior Mean','5% interval'),
         fill = cols, cex=0.65)
  axis(1, at=c(seq(0,1,length=12)),
       labels=c('Jan', 'Feb','Mar', 'Apr','May',
                'Jun','Jul', 'Aug', 'Sep', 'Oct','Nov', 'Dec'), cex=0.4)
dev.off()
# d
hot_day = data$time[which.max(predict(polynomial(betas_mean), data$time))]
hot_day
hot_day = as.numeric(betas_mean[2] / (-betas_mean[3]*2))
hot_day
hot_date = as.Date(hot_day*n, origin="2016-01-01")
hot_date

xGrid = seq(0,1,0.001)
a = dnorm(xGrid, mean=hot_day, sd=0.1)
tjo = max(a)
plot(as.Date(xGrid*n, origin="2016-01-01"),a, type='l')
abline(h=tjo/2)
```

Appendix B - Code for assignment 2

```
require('mvtnorm')
require('LaplacesDemon')
imgw = 5
imgh = 4
data = read.table('WomenWork.dat.txt', header=TRUE)
response = data$Work
features = as.matrix(data[,-data$Work])

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

# b
nFeats = ncol(features)
mu = rep(0, nFeats)
tau = 10
sigma2 = tau^2 * diag(nFeats)
beta_prior = rnorm(mu, sigma2)

logPostLogistic = function(betas, y, X, mu, sigma2) {
  nPara = length(betas)
  yPred = as.matrix(X) %*% betas

  logLike = sum(yPred*y - log(1 + exp(yPred)))
  if (abs(logLike) == Inf) {
    logLike = -20000
  }
  logPrior = dmvnorm(betas, rep(0, nPara), sigma2, log=TRUE)
  logPost = logPrior + logLike
  return (logPost)
}

initValues = rep(0, nFeats)
optimResults = optim(initValues,
                     logPostLogistic, y=response, X=features, mu=mu, sigma2=sigma2,
                     method=c('BFGS'), control=list(fnscale=-1), hessian=TRUE)

postMode = optimResults$par
postCov = -solve(optimResults$hessian)
approxPostStd = sqrt(diag(postCov))

names(postMode) = colnames(features)
names(approxPostStd) = colnames(features)

childMean = postMode['NSmallChild']
childStd = approxPostStd['NSmallChild']

betaGrid = seq(-abs(childMean - 4*childStd), abs(childMean + 4* childStd), length = 1000)

pdf('plots/smallchildren.pdf', width=imgw, height=imgh)
  plot(betaGrid, dnorm(betaGrid, childMean, childStd),
       type = "l", lwd = 2, ylab = '', xlab = expression(beta), col='steelblue')
```

```
  lines(qnorm(c(0.025, 0.975), childMean, childStd), c(0,0), lwd=2, col='tomato')
  legend('topright', legend=c('95% equal tail interval'), fill=c('tomato'), inset=0.02, cex=0.6)
dev.off()
# c

inverseLogit = function(betas, features){
  return (exp(features %*% betas) / (1 + exp(features %*% betas)))
}

jane_doe = c(1, 10, 8, 10, 1, 40, 1, 1)
betas = rmvnorm(1000, mean=postMode, sigma=postCov)

p = apply(betas, 1, inverseLogit, features=jane_doe)
y = sapply(p, rbern, n=1)

h = hist(y, breaks=2, plot=FALSE)
h$counts = h$counts / sum(h$counts)
h$counts = h$counts * 100
names(h$counts) = h$counts

pdf('plots/working.pdf', width=imgw, height=imgh)
  barplot(h$counts, ylab='Confidence', ylim=c(0,100), col=c('tomato', 'dodgerblue'),
          main='', legend=c('Not working', 'Working'))
dev.off()
```

Lab3 Report

Ludvig Noring & Michael Sörsäter

14 May 2017

Rainfall

a) Normal model

A Gibbs sampler is implemented that simulates from the joint posterior and after about 1500 iterations the mean and variance converges. The result can be shown in Figure 1 where 2500 iterations is used.

$$\mu|\sigma^2, x \sim N(\mu_n, \tau_n^2)$$
$$\sigma^2|\mu, x \sim Inv - \chi^2\left(v_n, \frac{v_0\sigma_0^2 + \sum_{i=1}^{n}(x_i - \mu)^2}{n + v_0}\right)$$

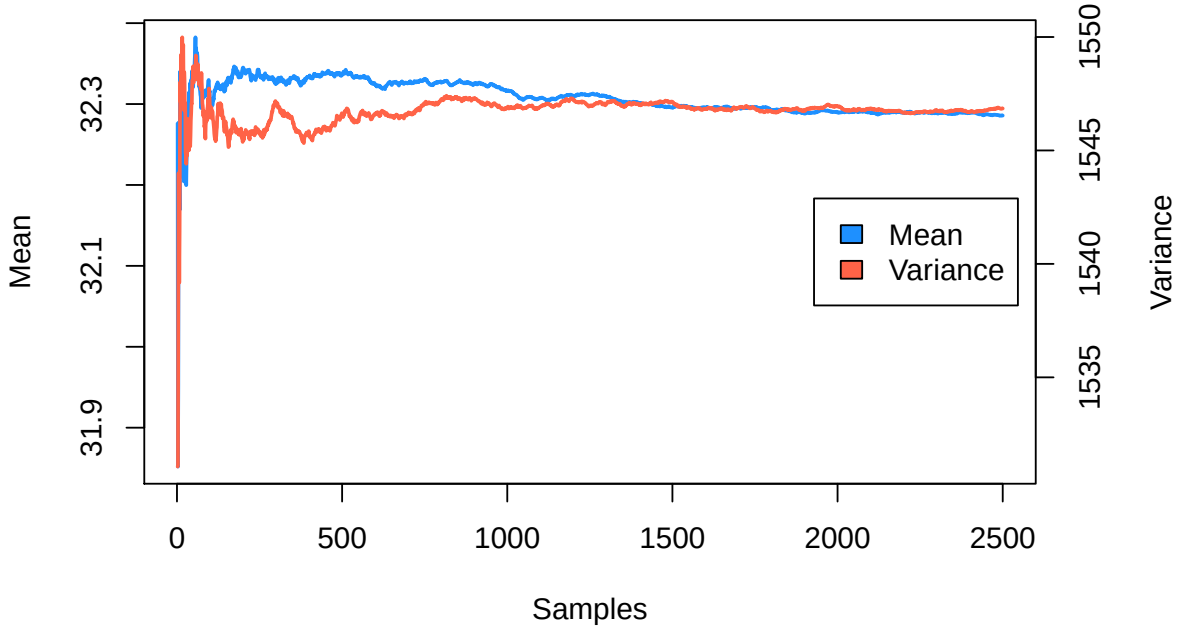

Figure 1: Convergence of the Mean

b) Mixture normal model

We use the provided code and modified it to suite our model. We updated the μ hyperparameters to:

μ_1 = mode of the density

μ_2 = mean of the data

The convergence of the mean and variance can be shown in Figure 2 and Figure 3.

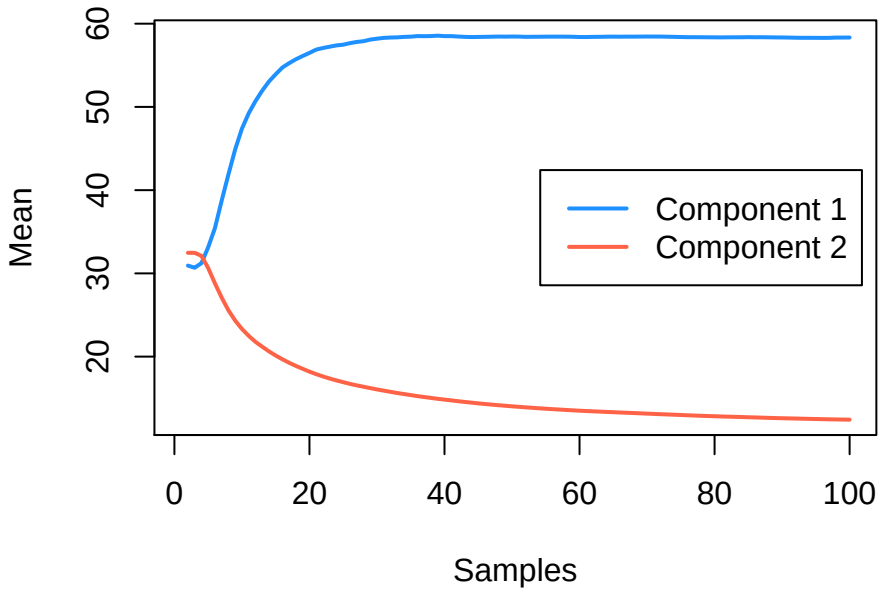

Figure 2: Convergence of the Mean

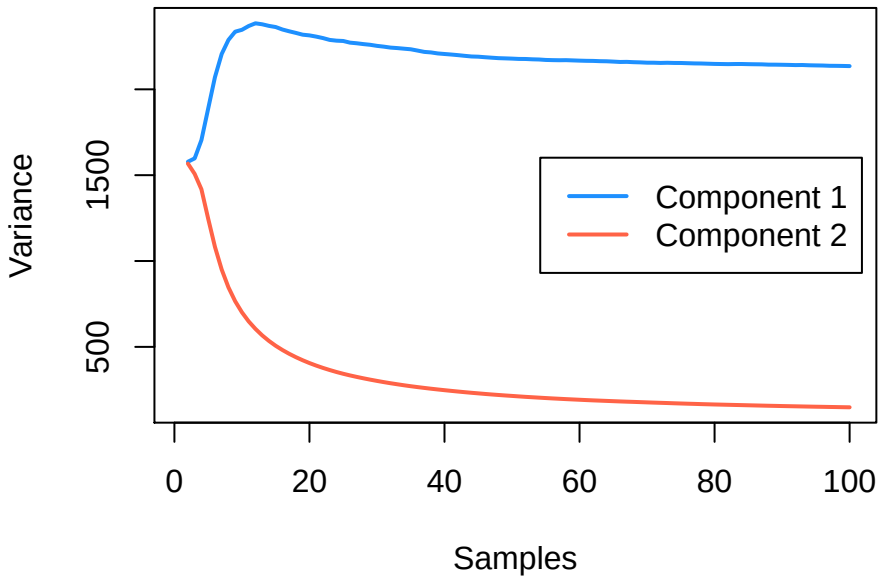

Figure 3: Convergence of the Variance

c)

The density from the original data together with the densities from a and b are shown in Figure 4. It is clear that the mixture model fits the data much more accurately.

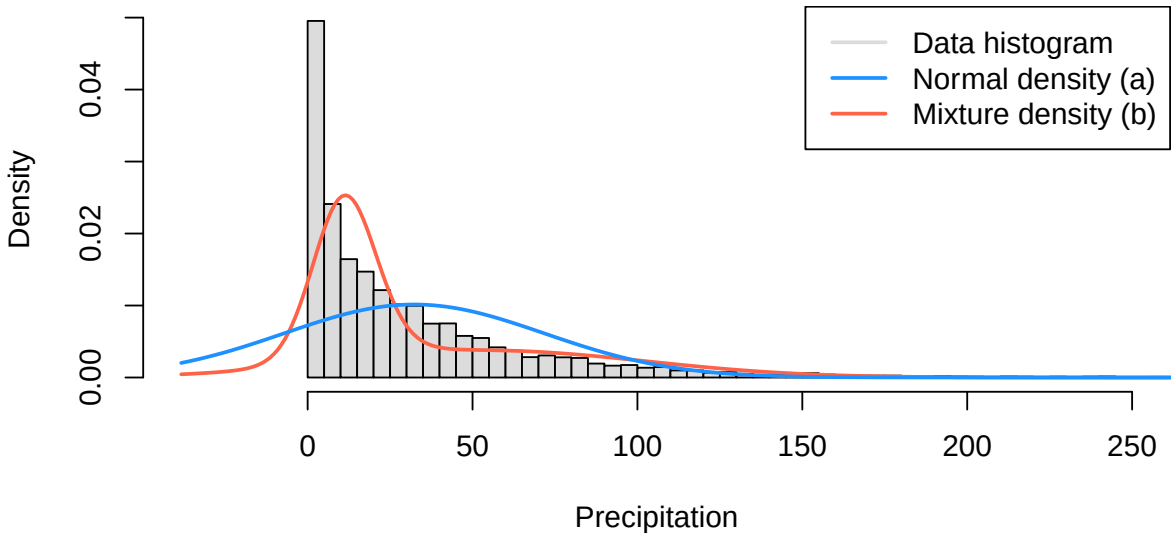

Figure 4: Mixture of normals

2 Probit regression

The Gibbs sampler for the probit regression model is implemented. With 10 000 iterations in the sampling the mean of the β values are calculated. To see how good these values fit the original data the confusion matrix is produced.

	0	1
0	65	35
1	32	68

Which have the misclassification rate 33.5 %.

By using optim to optimize β the following confusion matrix is produced.

	0	1
0	66	27
1	31	76

Which have the misclassification rate 29 %.

By plotting the histogram for each parameter together with the normal model that optim produce the following plot is derived. The histograms for the parameters resembles normal approximations pretty good. The result from the Gibbs sampling and optim are however different.

Some parameters have similar mean, for example HusbandInc and NBigChild but others are way off. The variance for Age and NSmallCHild are quite similar to the histograms.

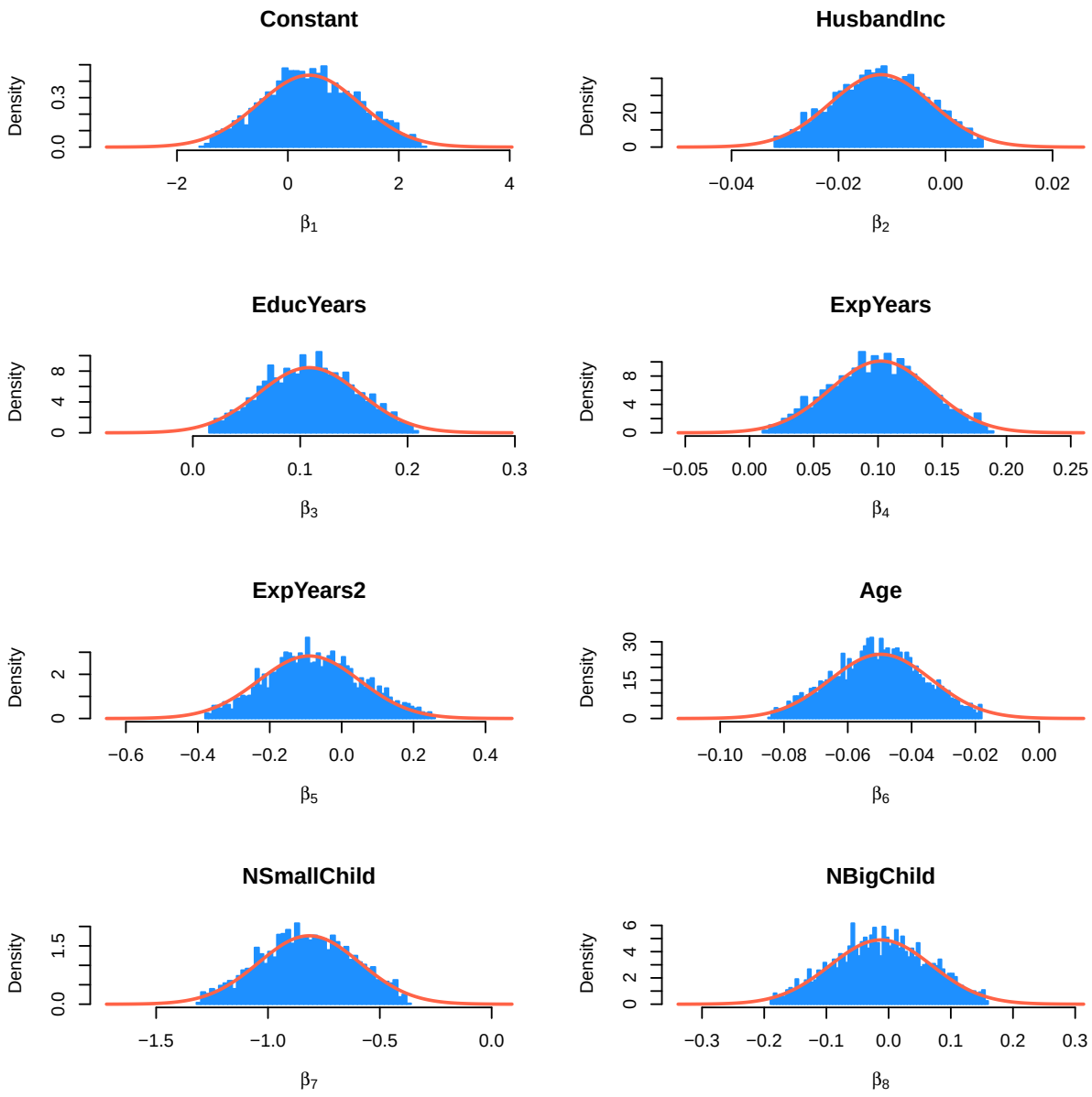

Figure 5: Histogram and density plots for β

Appendix A - Code for assignment 1

```
require('geoR')

imgw = 7
imgh = 4
rainfall = read.table('rainfall.dat')[,1]
set.seed(12345)
n = length(rainfall)

# a
mu_0 = mean(rainfall)
sigma2_0 = 1
v_0 = 1

nDraws = 2500

sigma2 = rinvchisq(n=1, v_0, sigma2_0)
# Lecture 7 slide 17
result = matrix(0, nDraws, 4)
for(i in 1:nDraws) {
  mu = rnorm(n=1, mean=mu_0, sd=sqrt(sigma2 / n))
  sigma2 = rinvchisq(n=1, df=(v_0 + n), scale=(v_0*sigma2_0 +  sum((rainfall - mu)^2)) / (n + v_0))
  result[i,] = c(mu, sigma2, mean(result[1:i-1,1]), mean(result[1:i-1,2]))
}

#pdf('plots/mu.pdf', width=imgw, height=imgh)
#plot(result[,3], type='l', xlab='Samples', ylab='Mean', col='dodgerblue')
#dev.off()
#pdf('plots/sigma.pdf', width=imgw, height=imgh)
#plot(result[,4], type='l', xlab='Samples', ylab='Variance', col='tomato')
#dev.off()

# Plot mean and variance in same plot
d = data.frame(x=seq(1, nDraws), mu=result[,3], sigma=result[,4])

pdf('plots/mean_variance.pdf', width=imgw, heigh=imgh)
par(mar = c(5,5,2,5))
with(d, plot(x, mu, type='l', col='dodgerblue', ylab='Mean', lwd=2, xlab='Samples'))
par(new = T)
with(d, plot(x, sigma, type='l', col='tomato', axes=F, xlab=NA, ylab=NA, lwd=2))
axis(side = 4)
mtext(side=4, line=3, 'Variance')
legend('right', legend=c('Mean', 'Variance'), fill=c('dodgerblue', 'tomato'), inset=0.02)
dev.off()

# b

# Using template for this part

# Estimating a simple mixture of normals
# Author: Mattias Villani, IDA, Linkoping University. http://mattiasvillani.com
```

```r
x = as.matrix(read.table('rainfall.dat')[,1])

# Model options
nComp = 2    # Number of mixture components

# Prior options
alpha = 10*rep(1,nComp) # Dirichlet(alpha)
muPrior = c(x[which.max(density(x)$y)], mean(x)) # Prior mean of theta
tau2Prior = rep(10,nComp) # Prior std theta
sigma2_0 = rep(var(x),nComp) # s20 (best guess of sigma2)
nu0 = rep(4,nComp) # degrees of freedom for prior on sigma2

# MCMC options
nIter = 100 # Number of Gibbs sampling draws

# Plotting options
plotFit = TRUE
lineColors = c("dodgerblue", "lawngreen", "magenta", 'yellow')
sleepTime = 0.02 # Adding sleep time between iterations for plotting
################   END USER INPUT ###############

###### Defining a function that simulates from the
rScaledInvChi2 = function(n, df, scale){
  return((df*scale)/rchisq(n,df=df))
}

####### Defining a function that simulates from a Dirichlet distribution
rDirichlet = function(param){
  nCat = length(param)
  thetaDraws = matrix(NA,nCat,1)
  for (j in 1:nCat){
    thetaDraws[j] = rgamma(1,param[j],1)
  }
  # Diving every column of ThetaDraws by the sum of the elements in that column.
  thetaDraws = thetaDraws/sum(thetaDraws)
  return(thetaDraws)
}

# Simple function that converts between two different representations of the mixture allocation
S2alloc = function(S){
  n = dim(S)[1]
  alloc = rep(0,n)
  for (i in 1:n){
    alloc[i] = which(S[i,] == 1)
  }
  return(alloc)
}

# Initial value for the MCMC
nObs = length(x)
# nObs-by-nComp matrix with component allocations.
S = t(rmultinom(nObs, size = 1 , prob = rep(1/nComp,nComp)))
theta = quantile(x, probs = seq(0,1,length = nComp))
```

```
sigma2 = rep(var(x),nComp)
probObsInComp = rep(NA, nComp)

# Setting up the plot
xGrid = seq(min(x)-1*apply(x,2,sd),max(x)+1*apply(x,2,sd),length = 500)
xGridMin = min(xGrid)
xGridMax = max(xGrid)
mixDensMean = rep(0,length(xGrid))
effIterCount = 0
ylim = c(0,1.33*max(hist(x, breaks=50)$density))

result_mu = matrix(0, nIter, 4)
result_sigma = matrix(0, nIter, 4)
for (k in 1:nIter){
  message(paste('Iteration number:',k))
  # Just a function that converts between different representations of the group allocations
  alloc = S2alloc(S)
  nAlloc = colSums(S)
  print(nAlloc)
  # Update components probabilities
  w = rDirichlet(alpha + nAlloc)

  # Update theta's
  for (j in 1:nComp){
    precPrior = 1/tau2Prior[j]
    precData = nAlloc[j]/sigma2[j]
    precPost = precPrior + precData
    wPrior = precPrior/precPost
    muPost = wPrior*muPrior + (1-wPrior)*mean(x[alloc == j])
    tau2Post = 1/precPost
    theta[j] = rnorm(1, mean = muPost, sd = sqrt(tau2Post))
  }
  result_mu[k,] = c(theta[1], theta[2], mean(result_mu[1:k-1,1]), mean(result_mu[1:k-1,2]))

  # Update sigma2's
  for (j in 1:nComp){
    sigma2[j] = rScaledInvChi2(1, df = nu0[j] + nAlloc[j],
              scale = (nu0[j]*sigma2_0[j] + sum((x[alloc == j] - theta[j])^2))/(nu0[j] + nAlloc[j]))
  }
  result_sigma[k,] = c(sigma2[1], sigma2[2], mean(result_sigma[1:k-1,1]), mean(result_sigma[1:k-1,2]))
  # Update allocation
  for (i in 1:nObs){
    for (j in 1:nComp){
      probObsInComp[j] = w[j]*dnorm(x[i], mean = theta[j], sd = sqrt(sigma2[j]))
    }
    S[i,] = t(rmultinom(1, size = 1 , prob = probObsInComp/sum(probObsInComp)))
  }

  # Printing the fitted density against data histogram
  if (plotFit && (k%%1 ==0)){
    effIterCount = effIterCount + 1
    hist(x, breaks = 50, freq = FALSE,
         xlim = c(xGridMin,250),
```

```
        ylim = ylim, col="gainsboro",
        main = paste("Iteration number",k))
    mixDens = rep(0,length(xGrid))
    components = c()
    for (j in 1:nComp){
      compDens = dnorm(xGrid,theta[j],sd = sqrt(sigma2[j]))
      mixDens = mixDens + w[j]*compDens
      lines(xGrid, compDens, type = "l", lwd = 2, col = lineColors[j])
      components[j] = paste("Component ",j)
    }
    mixDensMean = ((effIterCount-1)*mixDensMean + mixDens)/effIterCount

    lines(xGrid, mixDens, type = "l", lty = 2, lwd = 3, col = 'tomato')
    legend("topright", box.lty = 1, legend = c("Data histogram",components, 'Mixture'),
           col = c("black",lineColors[1:nComp], 'tomato'), lwd = 2)
    #Sys.sleep(sleepTime)
  }

}
pdf('plots/mixture.pdf', width=7, height=imgh)
  hist(x, breaks = 100, freq = FALSE,
     xlim = c(xGridMin,250), main = "",
     col='gainsboro',
     xlab='Precipitation')
  lines(xGrid, mixDensMean, type = "l", lwd = 2, col = "tomato")
  lines(xGrid,
      dnorm(xGrid, mean = result[nrow(result),3], sd = sqrt(result[nrow(result),4])),
      type = "l", lwd = 2, col = "dodgerblue")
  legend("topright", box.lty = 1,
       legend = c("Data histogram","Normal density (a)", "Mixture density (b)"),
       col=c("gainsboro","dodgerblue","tomato"), lwd = 2)
dev.off()

ylim_mu = c(min(result_mu[-1,3:4]), max(result_mu[-1,3:4]))
ylim_sigma = c(min(result_sigma[-1,3:4]), max(result_sigma[-1,3:4]))

# Plot the result
pdf('plots/muMixed.pdf', width=5, height=imgh)
  plot(result_mu[,3], type='l', xlab='Samples', ylab='Mean', col='dodgerblue',
     ylim=ylim_mu, lwd=2)
  lines(result_mu[,4], col='tomato', lwd=2)
  legend("right", box.lty = 1, inset=0.02,
       legend=c("Component 1","Component 2"),
       col=c("dodgerblue", "tomato"), lwd = 2)
dev.off()


# Plot the result but with traceplot
mcmc_mu = mcmc(result_mu[,1:2])
ylims = c(min(mcmc_mu), max(mcmc_mu))
pdf('plots/muMixed-trace.pdf', width=5, height=imgh)
traceplot(mcmc_mu[,1], type='l', xlab='Samples', ylab='Mean', col='dodgerblue',
     ylim=ylims,#ylim_mu,
```

```
     lwd=2)
lines(mcmc_mu[,2], col='tomato', lwd=2)
legend("right", box.lty = 1, inset=0.02,
       legend=c("Component 1","Component 2"),
       col=c("dodgerblue", "tomato"), lwd = 2)
dev.off()



pdf('plots/sigmaMixed.pdf', width=5, height=imgh)
  plot(result_sigma[,3], type='l', xlab='Samples', ylab='Variance', col='dodgerblue',
     ylim=ylim_sigma, lwd=2)
  lines(result_sigma[,4], col='tomato', lwd=2)
  legend("right", box.lty = 1, inset=0.02,
    legend=c("Component 1","Component 2"),
    col=c("dodgerblue", "tomato"), lwd = 2)

dev.off()
```

Appendix B - Code for assignment 2

```
require('msm')
require('mvtnorm')

data = read.table('WomenWork.dat', header=TRUE)
Y = data$Work
X = as.matrix(data[,-data$Work])
nObs = nrow(X)
nFeats = ncol(X)

# Calculate the accuracy of the beta values
performance = function(betas){
  y_hat = as.vector(X %*% betas)
  y_hat = sign(y_hat)
  y_hat[y_hat==-1] = 0

  print(table(y_hat, Y))
  print(mean(y_hat != Y))
}

uGenerator = function(betas) {
  curMean = X %*% t(betas)
  u = rep(0, nObs)
  for(i in 1:nObs){
    # Define bounds = (lower, upper)
    if(Y[i] == 0){
      bounds = c(-Inf, 0)
    }else{
      bounds = c(0, Inf)
    }
    u[i] = rtnorm(n=1, mean=curMean[i], sd=1, lower=bounds[1], upper=bounds[2])
  }
  return(u)
}

# From the given u, draw new beta-values
betaGenerator = function(u, sigma2=1) {
  # From lecture 5, slide 8
  XtX = t(X) %*% X
  beta_hat = solve(XtX) %*% t(X) %*% as.matrix(u)
  mu_n = solve(XtX + omega_0) %*% (XtX %*% beta_hat + omega_0 %*% mu_0)
  omega_n = XtX + omega_0

  beta = rmvnorm(n=1, mean=mu_n, sigma=(sigma2*solve(omega_n)))
  return (beta)
}

# Initial values
mu_0 = 0
tau = 10
mu_0 = rep(0, nFeats)
covar_0 = tau^2 * diag(nFeats)
```

```
omega_0 = solve(covar_0)

beta_prior = as.matrix(rnorm(n=nFeats, mean=mu_0, sd=sqrt(diag(omega_0))))
u = uGenerator(t(beta_prior))

# Necessary with 1000 iterations, draws more to get smoother histograms
draws = 2500
# One column for each beta parameter
result_beta = matrix(0, draws, nFeats)
for(i in 1:draws) {
  print(i)
  beta = betaGenerator(u)
  u = uGenerator(beta)
  result_beta[i,] = beta
}

performance(colMeans(result_beta))

# c

logPostProbit = function(betas, y, X) {
  yPred = as.matrix(X) %*% betas

  logLike = sum(y*pnorm(yPred, log.p = TRUE) + (1-y)*pnorm(yPred, log.p = TRUE, lower.tail = FALSE))
  logPrior = dmvnorm(betas, mu_0, covar_0, log=TRUE);

  # add the log prior and log-likelihood together to get log posterior
  return(logLike + logPrior)

}

initValues = rep(0, nFeats)
optimResults = optim(initValues,
                     logPostProbit, y=Y, X=X,
                     method=c('BFGS'), control=list(fnscale=-1), hessian=TRUE)

postMode = optimResults$par
postCov = -solve(optimResults$hessian)
approxPostStd = sqrt(diag(postCov))

performance(postMode)

pdf('plots/betas.pdf')
  par(mfrow=c(4,2))
  for(i in 1:nFeats) {
    mean = postMode[i]
    sd = approxPostStd[i]

    # Remove outliers. (2 < x < 98) %
    draws = result_beta[,i]
    threshold = 0.02
    bounds = quantile(draws, probs=c(threshold, 1 - threshold))
    draws = draws[draws > bounds[1]]
```

```
    trimmed = draws[draws < bounds[2]]

    # Toggle to include/exclude outliers
    values = result_beta[,i]
    values = trimmed

    xRange = c(values, mean + sd * 4, mean - sd * 4)
    betaGrid = seq(min(xRange), max(xRange), length=1000)

    gibbs_density = hist(values, breaks=50, plot=FALSE)
    approx_density = dnorm(betaGrid, mean=mean, sd=sd)

    yMax = max(c(approx_density, gibbs_density$density))

    plot(gibbs_density, freq=FALSE, col='dodgerblue', border='dodgerblue',
         xlim=c(min(betaGrid), max(betaGrid)),
         ylim=c(0, yMax),
         xlab=substitute(beta[idx], list(idx=i)),
         main=colnames(X)[i])
    lines(betaGrid, approx_density, col='tomato', lwd=2)
  }
dev.off()
```

Lab4 Report

Ludvig Noring, Michael Sörsäter

May 17, 2017

Poisson regression

a)

Maximum likelihood estimation is performed with the generalized model using the family poisson. The result can be shown in the following table.

	Const	PowerSeller	VerifyID	Sealed	Minblem	MajBlem	LargNeg	LogBook	MinBidShare
1	1.0724	-0.0205	-0.3945	0.4438	-0.0522	-0.2209	0.0707	-0.1207	-1.8941

Table 1: Coefficients for glm

The covarities that significantly influence the response are VerifyID, Sealed and MinBidShare.

b)

We implemented the log posterior method for the Poisson model. This produced similar results to what we got with glm. The result can be shown in the following table.

	Const	PowerSeller	VerifyID	Sealed	Minblem	MajBlem	LargNeg	LogBook	MinBidShare
1	1.0698	-0.0205	-0.3930	0.4436	-0.0525	-0.2212	0.0707	-0.1202	-1.8920

Table 2: Coefficients for optim

c)

We implemented a general random walk-metropolis alhorithm and used it with our log-Poission-Posterior function. With 1000 draws the theta converges as shown in Figure 1.

	Const	PowerSeller	VerifyID	Sealed	Minblem	MajBlem	LargNeg	LogBook	MinBidShare
1	1.0671	-0.0154	-0.4038	0.4414	-0.0506	-0.2268	0.0700	-0.1172	-1.8949

Table 3: Coefficients for optim

d)

Using the MCMC from above the probability for k number of bids are presented in Figure 2. As shown in the Figure, the probability of 0 bids are 35.6 %.

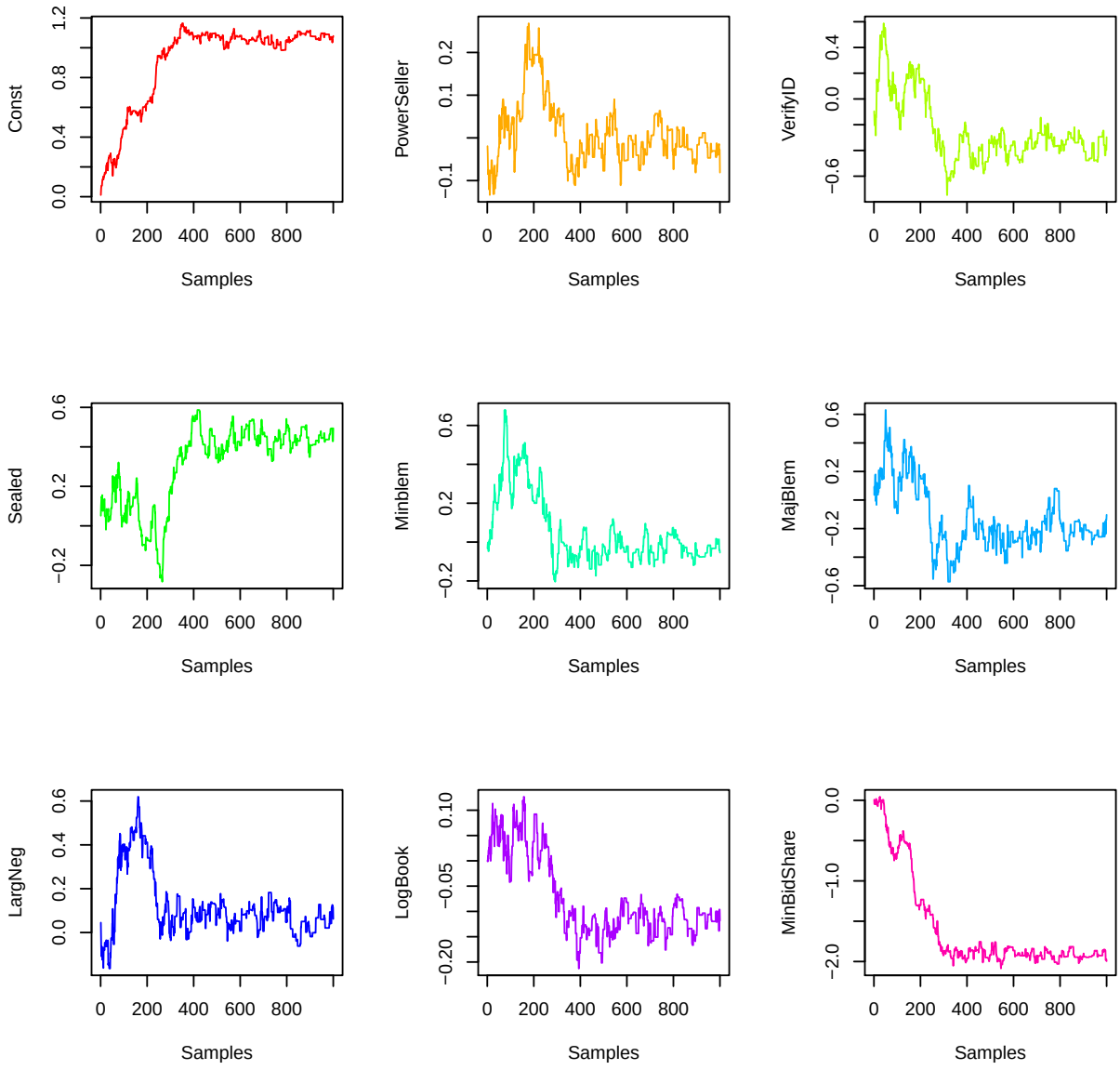

Figure 1: Convergence of θ

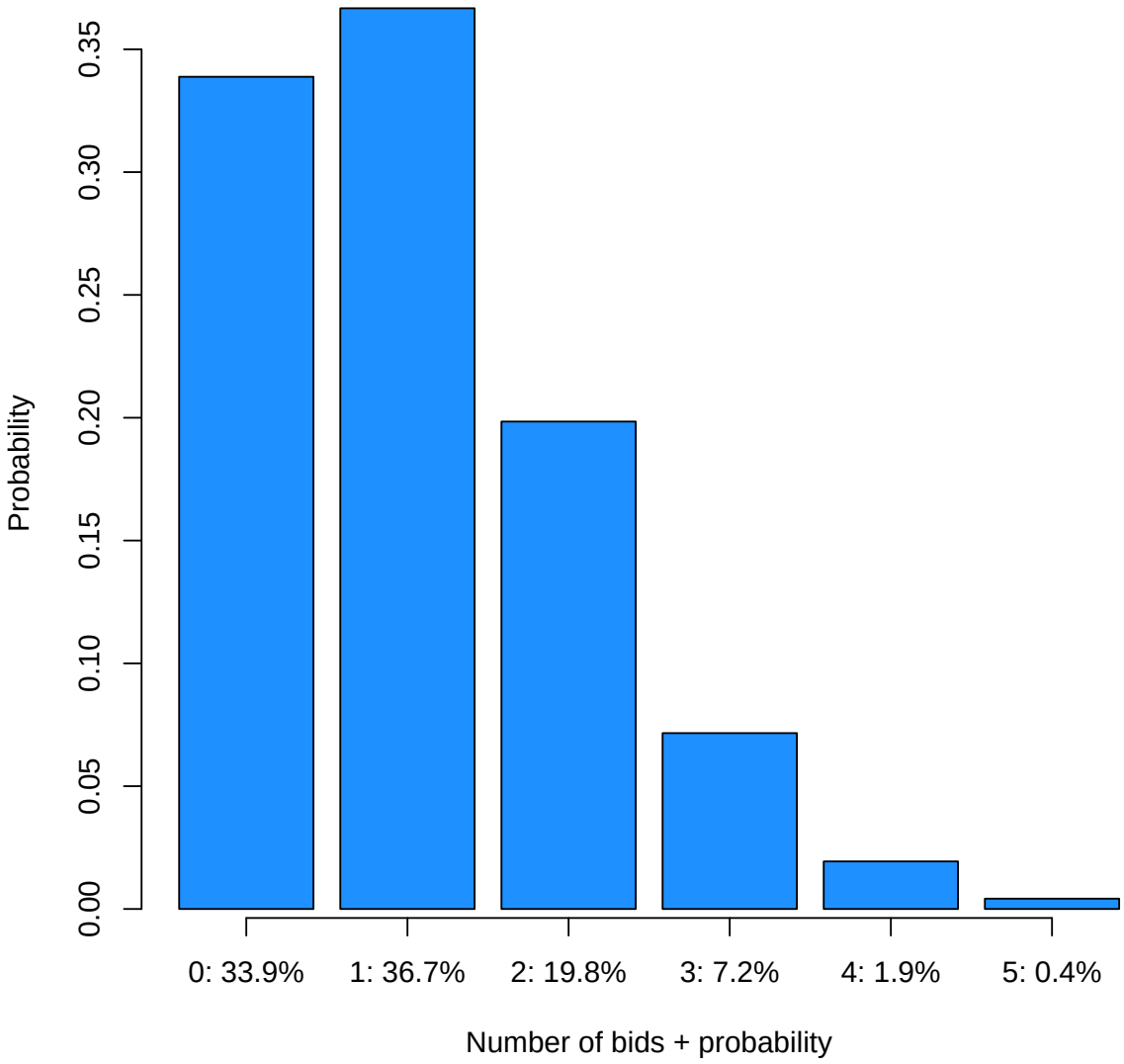

Figure 2: Bid Predictions

Appendix - Code for the assignment

```
require('MASS')
require('geoR')
require('mvtnorm')

ebay = read.table('eBayNumberOfBidderData.dat', header = TRUE)
X = as.matrix(ebay[,-1])
Y = ebay[,1]

nFeats = ncol(X)
nObs = nrow(X)

# a
glmModel = glm(nBids ~ 0 + ., data=ebay, family=poisson)
coefs = glmModel$coefficients
coefs

# b
sigma2_0 = 100 * solve(t(X) %*% X)
mu_0 = rep(0, nFeats)

beta_prior = mvrnorm(n=1, mu=mu_0, Sigma=sigma2_0)

# Calculate the posterior for the poisson model
logPostPoi = function(betas, y, X) {
  yPred = as.matrix(X) %*% betas
  lambda = exp(yPred)

  logLike = sum(log(exp(-lambda) * lambda^y / factorial(y)))
  logPrior = dmvnorm(betas, mu_0, sigma2_0, log=TRUE);

  # add the log prior and log-likelihood together to get log posterior
  return(logLike + logPrior)
}

initValues = rep(0, nFeats)
optimResults = optim(initValues,
                     logPostPoi, y=Y, X=X,
                     method=c('BFGS'), control=list(fnscale=-1), hessian=TRUE)

postMode = optimResults$par
postCov = -solve(optimResults$hessian)
approxPostStd = sqrt(diag(postCov)/nFeats)
postMode

# c
metroDraw = function(logPostFunc, thetas, sigma, c_tilde, ...) {
  draws = 1000
  result_theta = matrix(0, draws, nFeats)
  mean_conv = matrix(0, draws, nFeats)
  thetas_c = thetas
  prob = c()
```

```
  for(i in 1:draws){
    thetas_p = (mvrnorm(1, thetas_c, c_tilde*sigma))
    alpha = min(1, exp(logPostFunc(thetas_p, ...) - logPostFunc(thetas_c, ...)))
    prob = c(prob, alpha)
    # Update with probability alpha
    update = sample(c(TRUE, FALSE), 1, prob=c(alpha, 1-alpha))
    if(update)
      thetas_c = thetas_p

    result_theta[i,] = thetas_c

    if (i > 1)
      mean_conv[i,] = colMeans(result_theta[1:i,])
  }
  print(paste('Mean of alpha:', mean(prob)))
  return(result_theta)
 # return (mean_conv)
}

c_tilde = 0.6
thetas = metroDraw(logPostPoi, thetas=rep(0, length(beta_prior)), sigma=postCov,
                   c_tilde=c_tilde, y=Y, X=X)

pdf('plots/convergence.pdf')
  par(mfrow=c(3,3))
  for(i in 1:nFeats) {
    #plot(thetas[,i], type='l', col=rainbow(nFeats)[i], xlab='Samples', ylab=colnames(X)[i])
    traceplot(mcmc(thetas[,i]), col=rainbow(nFeats)[i], xlab='Samples', ylab=colnames(X)[i])
  }
dev.off()

# d
optTheta = thetas[nrow(thetas),]
femkrona_1972 = c(1, 1, 1, 1, 0, 0, 0, 1, 0.5)
names(femkrona_1972) = colnames(X)
lambda = exp(femkrona_1972%*%optTheta)

xGrid = 0:5
xAxis = paste(xGrid, ': ', round(100*dpois(xGrid, lambda), 1), '%', sep='')

pdf('plots/femkrona.pdf')
  barloc = barplot(dpois(xGrid, lambda), xaxt='n',
                   col='dodgerblue', xlab='Number of bids + probability', ylab='Probability')
  axis(side=1, at=barloc, labels = xAxis)
dev.off()
```