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.

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

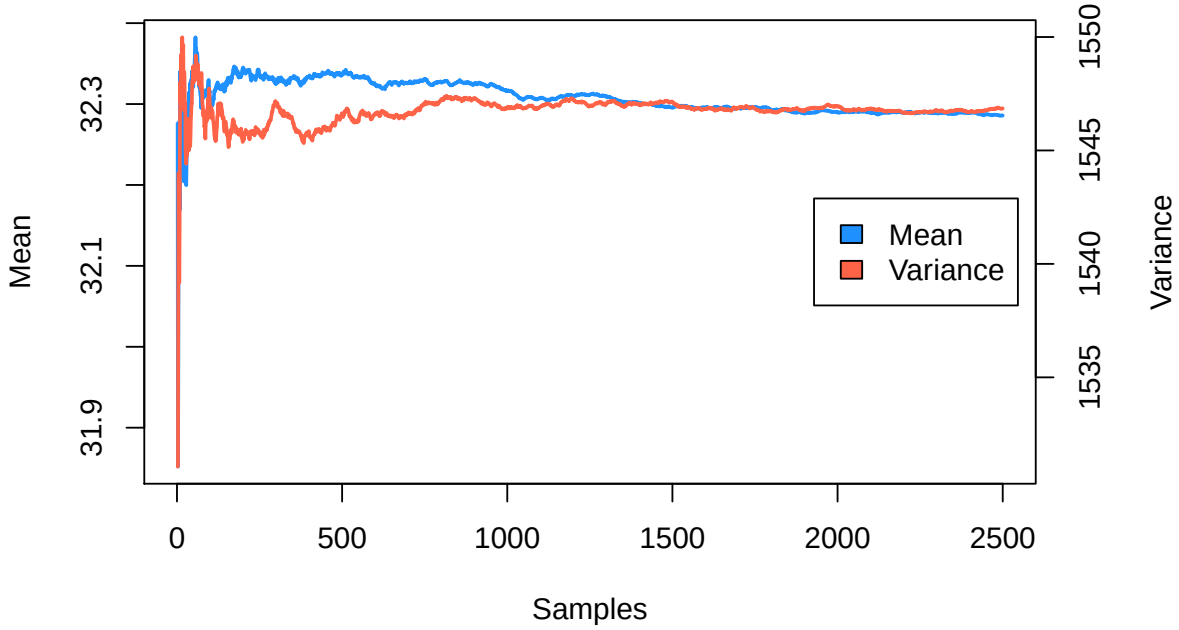

Figure 1: Convergence of the Mean

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.

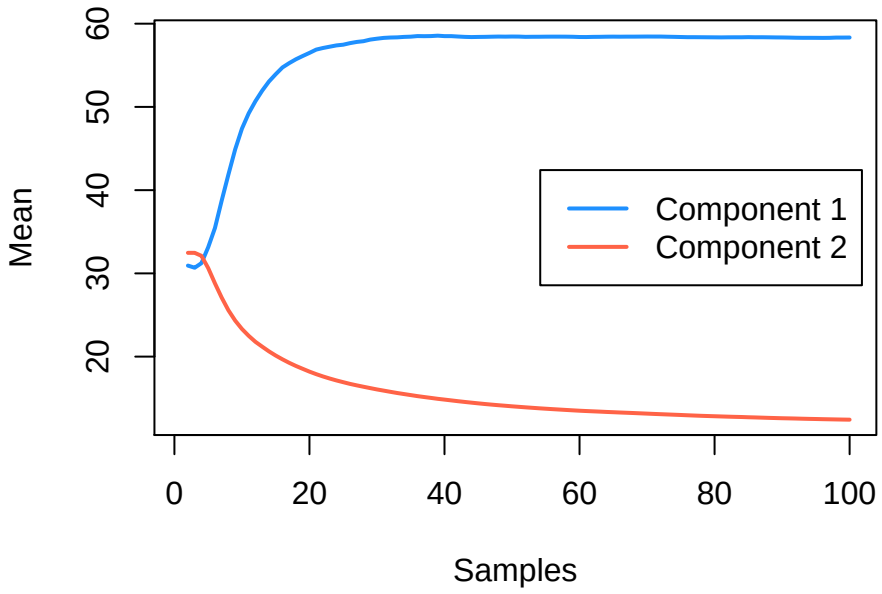

Figure 2: Convergence of the Mean

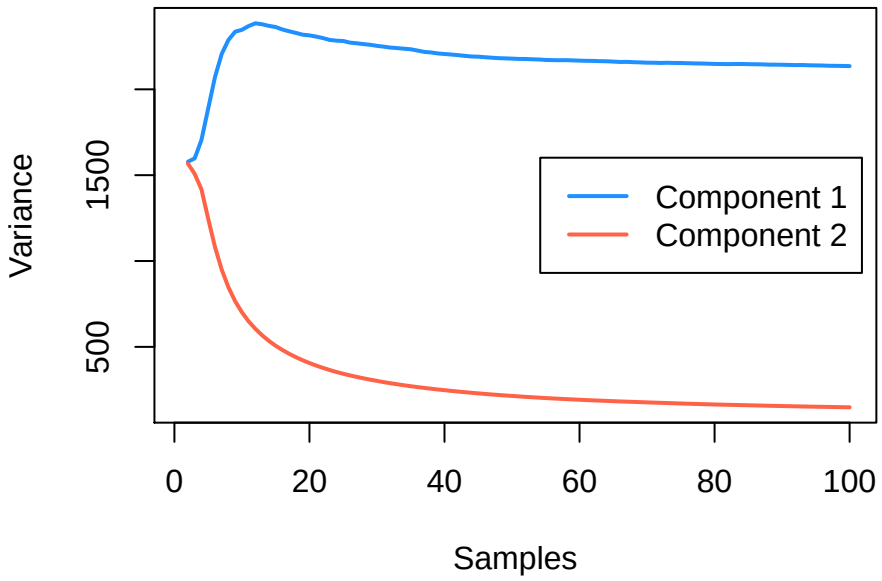

Figure 3: Convergence of the Variance

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.

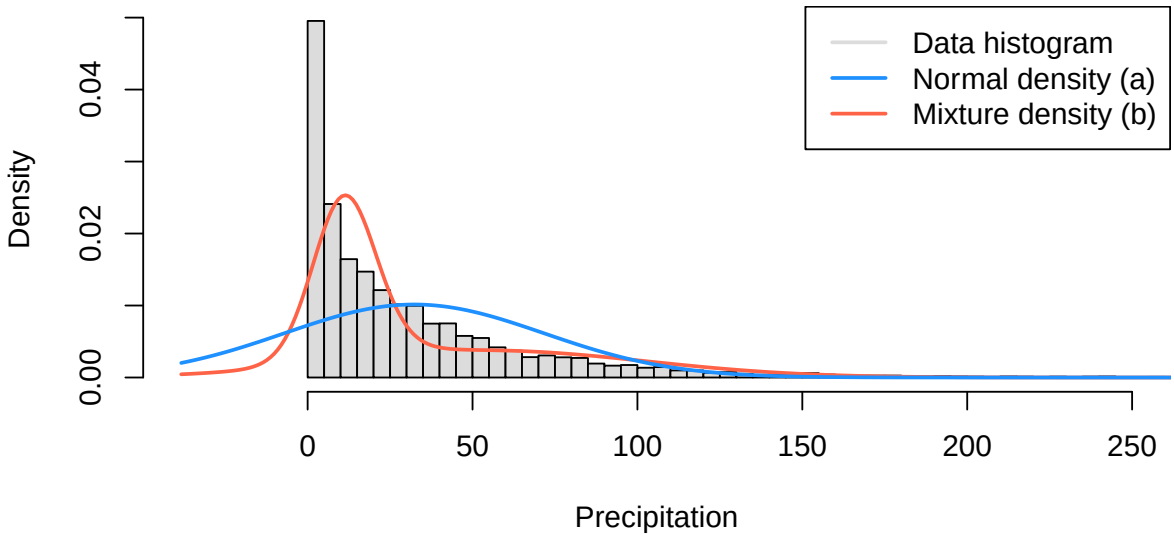

Figure 4: Mixture of normals

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.

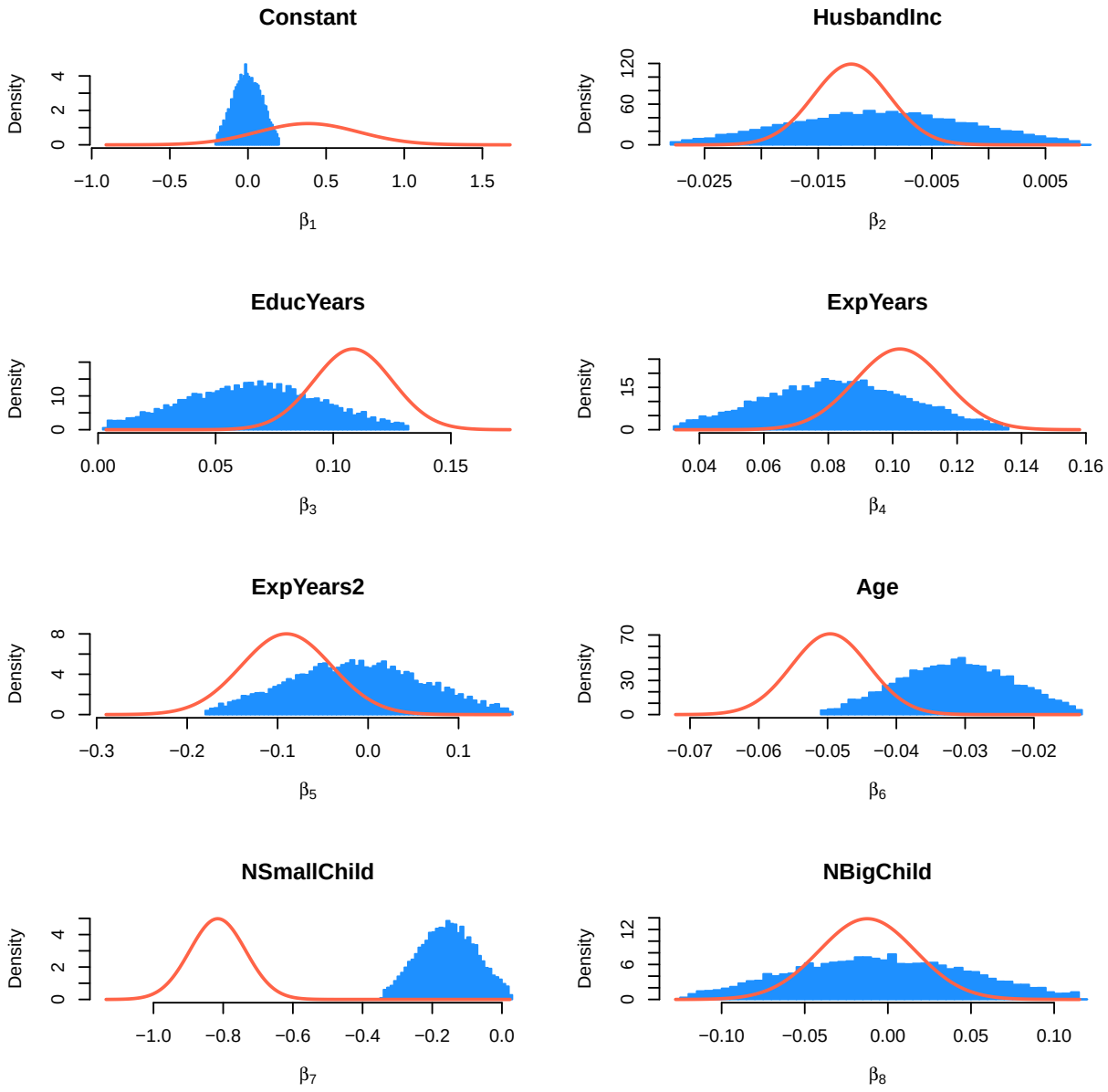

Figure 5: Histogram and density plots for β

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)
omega_0 = tau^2 * diag(nFeats)
```

```
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 = 500
# 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, omega_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)/nFeats)

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()
```