Exams

Christian von Koch

2020-05-31

#2017-05-30

## Assignment 1

## a) Plot the posterior distribution of theta  
  
riceData <- c(1.556, 1.861, 3.135, 1.311, 1.877, 0.622, 3.219, 0.768, 2.358, 2.056)  
  
# Random number generator for the Rice distribution  
rRice <-function(n = 1, theta = 1, psi = 1){  
 x <- rnorm(n = n, mean = 0, sd = sqrt(psi))  
 y <- rnorm(n = n, mean = theta, sd = sqrt(psi))  
 return(sqrt(x^2+y^2))  
}  
  
# Function for calculating the log posterior distrib with theta prior set to 1  
logPosterior = function(data, theta, psi) {  
 bessel\_factor=1  
 for (i in data) {  
 bessel\_factor=bessel\_factor\*besselI(i\*theta/psi, nu=0)  
 }  
 post=-log(psi)-1/(2\*psi)\*sum(data^2+theta^2)+log(bessel\_factor)  
 return(post+0) # If prior is assumed to be constant we set the prior to 1 which in log scale yields 0  
}  
  
gridWidth=0.01  
theta\_grid=seq(0,3,gridWidth)  
posterior\_distrib\_log=sapply(theta\_grid, logPosterior, data=riceData, psi=1)  
posterior\_distrib\_norm=1/gridWidth\*exp(posterior\_distrib\_log)/sum(exp(posterior\_distrib\_log))  
sum(posterior\_distrib\_norm)  
plot(theta\_grid, posterior\_distrib\_norm, xlab=expression(theta), ylab="Density", main="Posterior density of theta",  
 type="l", lwd=2)  
  
## b) Use numerical optimization to obtain a normal approx. of the posterior distrib of theta. Overlay curve   
## from a) with the approximated normal distribution  
  
# Defining initial values to be passed on to the optimizer  
set.seed(12345)  
initVal = rnorm(1, mean=0, sd=1)  
  
# Finding the optimized betavector  
optimResult = optim(initVal, logPosterior, data=riceData, psi=1, method=c("L-BFGS-B"),  
 control=list(fnscale=-1), lower=0, hessian=TRUE)  
  
# Defining the values of interest  
postMode = optimResult$par  
postCov = as.numeric(-solve(optimResult$hessian))  
print("The posterior mode is:")  
print(postMode)  
print("The approximated standard deviation is:")  
print(postCov)  
lines(theta\_grid, dnorm(theta\_grid, mean=postMode, sd=sqrt(postCov)), col="red", lwd=2)  
legend(x = 1.8, y = 1, legend = c("True posterior", "Approximate posterior"),   
 col = c("black","red"), lty = c(1,1), lwd = c(2,2), cex = 0.8)  
  
## Answer: Not perfect approx but fairly good.   
  
## c) Simulate distrib for new observation using normal approx in b)  
  
nDraws=5000  
set.seed(12345)  
theta=rnorm(nDraws, mean=postMode, sd=sqrt(postCov))  
pred\_distrib=c()  
for (i in theta) {  
 pred\_distrib=c(pred\_distrib, rRice(theta=i))  
}  
  
hist(pred\_distrib, breaks=100, xlab="Index", main="Predictive density of new obs")

##Assignment 2

## a) Model posterior data with prior Gamma and likelihood Poisson, plot the posterior  
  
# We know that posterior mapping with gamma prior and poisson likelihood is gamma distributed  
  
sumBids=sum(bids)  
n=length(bids)  
alpha=1  
beta=1  
posterior\_theta=dgamma(seq(3,4,0.001), alpha+sumBids, beta+n)  
plot(seq(3,4,0.001), posterior\_theta, type="l", lwd=2)  
  
# b) Investigate through graphical methods if Poisson model describes data well  
  
xGrid=seq(min(bids), max(bids))  
data\_norm=bidsCounts/sum(bidsCounts)  
nDraws=5000  
thetaDraws=rgamma(nDraws, alpha+sumBids, beta+n)  
poissonDensity=rep(0, length(xGrid))  
for (i in thetaDraws) {  
 poissonDensity=poissonDensity+dpois(xGrid, lambda=i)  
}  
  
avgPoissonDensity=poissonDensity/nDraws  
plot(xGrid, data\_norm, xlab="No. of bids", ylab="Density", main="Fitted models", type="o", cex=0.8,  
 ylim=c(0,0.25), lwd=2)  
lines(xGrid, avgPoissonDensity, col="red", lwd=2, type="o")  
legend(x=7, y=0.2, col=c("black", "red"), legend=c("Data", "Poisson mean density"), lty=c(1,1),   
 lwd=c(2,2), pch=c("o", "o"))  
  
## Terrible fit which the plot shows  
  
## c) Use GibbsMixPois.R. Esimate the mixture of Poissons both with K=2 and K=3. nIter=5000.  
  
GibbsMixPois <- function(x, nComp, alpha, alphaGamma, betaGamma, xGrid, nIter){  
   
 # Gibbs sampling for a mixture of Poissons  
 # Author: Mattias Villani, IDA, Linkoping University. http://mattiasvillani.com  
 #  
 # INPUTS:  
 # x - vector with data observations (counts)  
 # nComp - Number of mixture components to be fitted  
 # alpha - The prior on the mixture component weights is w ~ Dir(alpha, alpha,..., alpha)   
 # alphaGamma and betaGamma -   
 # The prior on the mean (theta) of the Poisson mixture components is   
 # theta ~ Gamma(alphaGamma, betaGamma) [rate parametrization of the Gamma dist]  
 # xGrid - the grid of data values over which the mixture is evaluated and plotted  
 # nIter - Number of Gibbs iterations  
 #  
 # OUTPUTS:  
 # results$wSample - Gibbs sample of mixture component weights. nIter-by-nComp matrix  
 # results$thetaSample - Gibbs sample of mixture component means. nIter-by-nComp matrix  
 # results$mixDensMean - Posterior mean of the estimated mixture density over xGrid.  
   
   
 ####### 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)  
 }  
 thetaDraws = thetaDraws/sum(thetaDraws) # Diving every column of ThetaDraws by the sum of the elements in that column.  
 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 values for the Gibbs sampling  
 nObs <- length(x)  
 S <- t(rmultinom(nObs, size = 1 , prob = rep(1/nComp,nComp))) # nObs-by-nComp matrix with component allocations.  
 theta <- rep(mean(x), nComp) # Each component is initialized at the mean of the data  
   
 # Setting up the grid where the mixture density is evaluated.  
 mixDensMean <- rep(0,length(xGrid))  
 effIterCount <- 0  
   
 # Setting up matrices to store the draws  
 wSample <- matrix(0, nIter, nComp)  
 thetaSample <- matrix(0, nIter, nComp)  
 probObsInComp <- rep(NA, nComp)  
   
 # Setting up the priors - the same prior for all components  
 alpha <- rep(alpha, nComp)   
 alphaGamma <- rep(alphaGamma, nComp)   
 betaGamma <- rep(betaGamma, nComp)   
   
 # HERE STARTS THE ACTUAL GIBBS SAMPLING  
   
 for (k in 1:nIter){  
 message(paste('Iteration number:',k))  
 alloc <- S2alloc(S) # Function that converts between different representations of the group allocations  
 nAlloc <- colSums(S)  
   
 # Step 1 - Update components probabilities  
 w <- rDirichlet(alpha + nAlloc)  
 wSample[k,] <- w  
   
 # Step 2 - Update theta's in Poisson components  
 for (j in 1:nComp){  
 theta[j] <- rgamma(1, shape = alphaGamma + sum(x[alloc == j]), rate = betaGamma + nAlloc[j])  
 }  
 thetaSample[k,] <- theta  
   
 # Step 3 - Update allocation  
 for (i in 1:nObs){  
 for (j in 1:nComp){  
 probObsInComp[j] <- w[j]\*dpois(x[i], lambda = theta[j])  
 }  
 S[i,] <- t(rmultinom(1, size = 1 , prob = probObsInComp/sum(probObsInComp)))  
 }  
   
 # Computing the mixture density at the current parameters, and averaging that over draws.  
 effIterCount <- effIterCount + 1  
 mixDens <- rep(0,length(xGrid))  
 for (j in 1:nComp){  
 compDens <- dpois(xGrid, lambda = theta[j])  
 mixDens <- mixDens + w[j]\*compDens  
 }  
 mixDensMean <- ((effIterCount-1)\*mixDensMean + mixDens)/effIterCount  
 }  
 return(results = list(wSample = wSample, thetaSample = thetaSample, mixDensMean = mixDensMean))  
}  
  
result\_comp2=GibbsMixPois(bids, nComp=2, alpha=1, alphaGamma = alpha, betaGamma = beta,   
 xGrid=xGrid, nIter=500)  
result\_comp3=GibbsMixPois(bids, nComp=3, alpha=1, alphaGamma = alpha, betaGamma = beta,   
 xGrid=xGrid, nIter=500)  
  
## c) Use graphical methods to investigate if mixture of poissons fits data well. Is K=2 enough or should we  
## use K=3?  
  
plot(xGrid, data\_norm, xlab="No. of bids", ylab="Density", main="Fitted models", type="o",  
 ylim=c(0,0.25), lwd=2)  
lines(xGrid, result\_comp2$mixDensMean, col="red", lwd=2, type="o")  
lines(xGrid, result\_comp3$mixDensMean, col="gray", lwd=2, type="o")  
legend(x=7, y=0.2, col=c("black", "red", "gray"),   
 legend=c("Data", "Mixture density with 2 components", "Mixture density with 3 components"),   
 lty=c(1,1,1), lwd=c(2,2, 2), pch=c("o", "o", "o"), cex=1)  
  
## Good enough with 2 components in the mixture density

##Assignment 3

###############################  
########## Problem 3 ##########   
###############################   
  
# Reading the cars data from file  
load("cars.RData")  
  
library(mvtnorm)  
  
# Defining a function that simulates from the scaled inverse Chi-square distribution  
rScaledInvChi2 <- function(n, df, scale){  
 return((df\*scale)/rchisq(n,df=df))  
}  
  
BayesLinReg <- function(y, X, mu\_0, Omega\_0, v\_0, sigma2\_0, nIter){  
 # Direct sampling from a Gaussian linear regression with conjugate prior:  
 #  
 # beta | sigma2 ~ N(mu\_0, sigma2\*inv(Omega\_0))  
 # sigma2 ~ Inv-Chi2(v\_0,sigma2\_0)  
 #   
 # Author: Mattias Villani, IDA, Linkoping University. http://mattiasvillani.com  
 #  
 # INPUTS:  
 # y - n-by-1 vector with response data observations  
 # X - n-by-nCovs matrix with covariates, first column should be ones if you want an intercept.  
 # mu\_0 - prior mean for beta  
 # Omega\_0 - prior precision matrix for beta  
 # v\_0 - degrees of freedom in the prior for sigma2  
 # sigma2\_0 - location ("best guess") in the prior for sigma2  
 # nIter - Number of samples from the posterior (iterations)  
 #  
 # OUTPUTS:  
 # results$betaSample - Posterior sample of beta. nIter-by-nCovs matrix  
 # results$sigma2Sample - Posterior sample of sigma2. nIter-by-1 vector  
   
 # Compute posterior hyperparameters  
 n = length(y) # Number of observations  
 nCovs = dim(X)[2] # Number of covariates  
 XX = t(X)%\*%X  
 betaHat <- solve(XX,t(X)%\*%y)  
 Omega\_n = XX + Omega\_0  
 mu\_n = solve(Omega\_n,XX%\*%betaHat+Omega\_0%\*%mu\_0)  
 v\_n = v\_0 + n  
 sigma2\_n = as.numeric((v\_0\*sigma2\_0 + ( t(y)%\*%y + t(mu\_0)%\*%Omega\_0%\*%mu\_0 - t(mu\_n)%\*%Omega\_n%\*%mu\_n))/v\_n)  
 invOmega\_n = solve(Omega\_n)  
   
 # The actual sampling  
 sigma2Sample = rep(NA, nIter)  
 betaSample = matrix(NA, nIter, nCovs)  
 for (i in 1:nIter){  
   
 # Simulate from p(sigma2 | y, X)  
 sigma2 = rScaledInvChi2(n=1, df = v\_n, scale = sigma2\_n)  
 sigma2Sample[i] = sigma2  
   
 # Simulate from p(beta | sigma2, y, X)  
 beta\_ = rmvnorm(n=1, mean = mu\_n, sigma = sigma2\*invOmega\_n)  
 betaSample[i,] = beta\_  
   
 }  
 return(results = list(sigma2Sample = sigma2Sample, betaSample=betaSample))  
}  
  
## a) Linear regression problem with given dataset. Use Mattias function to derive joint posterior.   
## i) Plot marginal distributions of each param  
## ii) Compute point estimates for each regression coefficient assuming loss function  
## iii) Construct 95 % equal tail probability intervals for each parameter and interpret them  
  
  
y=cars$mpg  
x=as.matrix(cars[2:ncol(cars)])  
mu\_0=c(0,0,0,0)  
omega\_0=0.01\*diag(x=4)  
nu\_0=1  
sigma\_sq\_0=36  
jointPostDistrib=BayesLinReg(y, x, mu\_0, omega\_0, nu\_0, sigma\_sq\_0, 1000)  
hist(jointPostDistrib$sigma2Sample, breaks=10, main=paste("Marginal distribution of", expression(sigma^2)),  
 xlab=expression(sigma^2))  
par(mfrow=c(2,2))  
for(i in 1:4) {  
 hist(jointPostDistrib$betaSample[,i], breaks=10, main=paste("Marginal distribution of ", expression(beta), i,   
 sep=""), xlab=paste(expression(beta),i, sep=""))  
}  
title("Marginal distributions of the different betavalues", line=-1, outer=TRUE)  
par(mfrow=c(1,1))  
  
# Linear loss function is posterior median  
median(jointPostDistrib$sigma2Sample)  
median(jointPostDistrib$betaSample[,1])  
median(jointPostDistrib$betaSample[,2])  
median(jointPostDistrib$betaSample[,3])  
median(jointPostDistrib$betaSample[,4])  
  
# Prediction intervals for each param  
quantile(jointPostDistrib$sigma2Sample, c(0.025, 0.975))  
quantile(jointPostDistrib$betaSample[,1], c(0.025, 0.975))  
quantile(jointPostDistrib$betaSample[,2], c(0.025, 0.975))  
quantile(jointPostDistrib$betaSample[,3], c(0.025, 0.975))  
quantile(jointPostDistrib$betaSample[,4], c(0.025, 0.975))  
  
## Answer: Interpretation of the credible interval for weight [-4.759964, -1.531457]. A one unit increase of weight  
## lowers the amount of miles per gallon between -4.759964 and -1.531457 with 95 % posterior probability.   
  
## b) Investigate if effect on mpg is different in cars with six cylinders compared to cars with 8 cylinders  
  
hist(jointPostDistrib$betaSample[,4]-jointPostDistrib$betaSample[,3], 50)  
quantile(jointPostDistrib$betaSample[,4]-jointPostDistrib$betaSample[,3], c(0.025, 0.975))  
  
## Answer: Since 0 is present in interval we can not say that there is a difference between 8 and 6 cylinders  
## with 95 % posterior probability.  
  
## c) Compute by simulation predictive distrib for a new car 4 cylinders and weight=3.5  
  
new\_x=c(1,3.5,0,0)  
pred\_y=rep(0,nIter)  
for (i in 1:nIter) {  
 pred\_y[i]=sum(new\_x\*jointPostDistrib$betaSample[i,])+rnorm(1,sd=sqrt(jointPostDistrib$sigma2Sample[i]))  
}  
hist(pred\_y, breaks=40, freq=FALSE)

##Assignment 4

## Maximizing posterior expected utility.  
  
post\_dens = function(x) {  
 return(gamma(6+x)/gamma(13+x))  
}  
  
barplot(post\_dens(seq(0,10,1)), type="l")  
  
# x6=10 seems to yield a low enough probability to be an upper bound for sum  
  
posterior\_prob=post\_dens(seq(0,10))  
posterior\_prob=posterior\_prob/sum(posterior\_prob)  
exp\_util=c()  
for (k in 0:10) {  
 exp\_util=c(exp\_util,(2^k-3))  
}  
  
exp\_post\_dens=sum(posterior\_prob\*exp\_util)  
print(exp\_post\_dens)

#2017-08-16

##Assignment 1

## a) Plot posterior density of theta, with normal prior and cauchy distrib as likelihood  
  
# Reading the data vector yVect from file  
load(file = 'CauchyData.RData')  
cauchydata=yVect  
  
dCauchy <- function(x, theta = 0, gamma = 1){  
 return(dens = (1/(pi\*gamma))\*(1/(1+((x-theta)/gamma)^2)))  
}  
  
dlognormal <- function(x, mu, sigma2){  
 return(dens = (1/(sqrt(2\*pi\*sigma2)\*x))\*exp((-1/(2\*sigma2))\*(log(x)-mu)^2))  
}  
  
logPrior\_theta = function(theta, mu, sigma\_sq) {  
 return(dnorm(theta, mean=mu, sd=sqrt(sigma\_sq), log=TRUE))  
}  
  
logPosterior = function(data, mu, sigma\_sq, theta=0, gamma=1) {  
 prior=logPrior(theta, mu, sigma\_sq)  
 likelihood=dCauchy(data, theta, gamma)  
 likelihood=sum(log(likelihood))  
 return(likelihood + prior)  
}  
  
mu=0  
sigma\_sq=100  
gamma=1  
gridWidth=0.01  
theta\_grid=seq(0,8,gridWidth)  
posterior\_distrib=sapply(theta\_grid, logPosterior, data=cauchydata, mu=mu, sigma\_sq=sigma\_sq, gamma=1)  
posterior\_distrib=1/gridWidth\*exp(posterior\_distrib)/sum(exp(posterior\_distrib))  
plot(theta\_grid, posterior\_distrib, type="l", lwd=2, main="Posterior density for theta", xlab=expression(theta),  
 ylab="Density")  
  
## b) gamma is unknown with prior lognormal.   
  
set.seed(12345)  
initVal = c(0,0)  
  
logJointPosterior = function(joint, data, mu, sigma\_sq) {  
 prior\_theta=logPrior(joint[1], mu, sigma\_sq)  
 prior\_gamma=log(dlognormal(joint[2], mu, sigma\_sq))  
 likelihood=dCauchy(data, joint[1], joint[2])  
 likelihood=sum(log(likelihood))  
 return(likelihood + prior\_theta + prior\_gamma)  
}  
  
# Finding the optimized theta and gamma  
optimResult = optim(initVal, logJointPosterior, data=cauchydata, mu=mu, sigma\_sq=sigma\_sq, method=c("L-BFGS-B"),  
 control=list(fnscale=-1), lower=c(-Inf, 0.001), upper=c(Inf, Inf), hessian=TRUE)  
  
# Defining the values of interest  
postMode = optimResult$par  
postCov = -solve(optimResult$hessian)  
names(postMode)=c("Theta", "Gamma")  
print("The posterior mode is:")  
print(postMode)  
print("The approximated standard deviation is:")  
print(postCov)  
  
## c) Use normal approx in 1b) to obtain marginal posterior for the 99 % percentile of the caucy distrib  
## theta + gamma \* tan(pi(0.99-0.5))  
  
library(rmvnorm)  
normal\_approx=rmvnorm(5000, mean=postMode, sigma=postCov)  
cauchy\_distrib=normal\_approx[,1]+normal\_approx[,2]\*tan(pi\*(0.99-0.5))  
hist(cauchy\_distrib, breaks=50, main="Marginal distribution of special case of caucby", xlab="Function value")

##Assignment 2

# Reading the data from file  
library(MASS)  
BostonHousing = Boston  
y = BostonHousing$medv  
X = cbind(1,BostonHousing[,1:13]) # Adding a column of ones for the intercept  
names(X)[1] <- "intercept"  
covNames <- names(X)  
y <- as.numeric(y)  
X <- as.matrix(X)  
  
library(mvtnorm)  
  
# Defining a function that simulates from the scaled inverse Chi-square distribution  
rScaledInvChi2 <- function(n, df, scale){  
 return((df\*scale)/rchisq(n,df=df))  
}  
  
BayesLinReg <- function(y, X, mu\_0, Omega\_0, v\_0, sigma2\_0, nIter){  
 # Direct sampling from a Gaussian linear regression with conjugate prior:  
 #  
 # beta | sigma2 ~ N(mu\_0, sigma2\*inv(Omega\_0))  
 # sigma2 ~ Inv-Chi2(v\_0,sigma2\_0)  
 #   
 # Author: Mattias Villani, IDA, Linkoping University. http://mattiasvillani.com  
 #  
 # INPUTS:  
 # y - n-by-1 vector with response data observations  
 # X - n-by-nCovs matrix with covariates, first column should be ones if you want an intercept.  
 # mu\_0 - prior mean for beta  
 # Omega\_0 - prior precision matrix for beta  
 # v\_0 - degrees of freedom in the prior for sigma2  
 # sigma2\_0 - location ("best guess") in the prior for sigma2  
 # nIter - Number of samples from the posterior (iterations)  
 #  
 # OUTPUTS:  
 # results$betaSample - Posterior sample of beta. nIter-by-nCovs matrix  
 # results$sigma2Sample - Posterior sample of sigma2. nIter-by-1 vector  
   
 # Compute posterior hyperparameters  
 n = length(y) # Number of observations  
 nCovs = dim(X)[2] # Number of covariates  
 XX = t(X)%\*%X  
 betaHat <- solve(XX,t(X)%\*%y)  
 Omega\_n = XX + Omega\_0  
 mu\_n = solve(Omega\_n,XX%\*%betaHat+Omega\_0%\*%mu\_0)  
 v\_n = v\_0 + n  
 sigma2\_n = as.numeric((v\_0\*sigma2\_0 + ( t(y)%\*%y + t(mu\_0)%\*%Omega\_0%\*%mu\_0 - t(mu\_n)%\*%Omega\_n%\*%mu\_n))/v\_n)  
 invOmega\_n = solve(Omega\_n)  
   
 # The actual sampling  
 sigma2Sample = rep(NA, nIter)  
 betaSample = matrix(NA, nIter, nCovs)  
 for (i in 1:nIter){  
   
 # Simulate from p(sigma2 | y, X)  
 sigma2 = rScaledInvChi2(n=1, df = v\_n, scale = sigma2\_n)  
 sigma2Sample[i] = sigma2  
   
 # Simulate from p(beta | sigma2, y, X)  
 beta\_ = rmvnorm(n=1, mean = mu\_n, sigma = sigma2\*invOmega\_n)  
 betaSample[i,] = beta\_  
   
 }  
 return(results = list(sigma2Sample = sigma2Sample, betaSample=betaSample))  
}  
  
mu\_0=rep(0, ncol(X))  
omega\_0=0.01\*diag(ncol(X))  
v\_0=1  
sigma2\_0=36  
nIter=5000  
  
bayes\_lin\_results=BayesLinReg(y, X, mu\_0, omega\_0, v\_0, sigma2\_0, nIter)  
# Under quadratic loss, posterior mean is point estimate  
beta\_estimates=rep(0,ncol(X))  
beta\_credIntervals=matrix(0, ncol(X), 2)  
for (i in 1:ncol(X)) {  
 beta\_estimates[i]=mean(bayes\_lin\_results$betaSample[,i])  
 beta\_credIntervals[i,]=quantile(bayes\_lin\_results$betaSample[,i], c(0.025, 0.975))  
}  
sigma\_estimate=mean(bayes\_lin\_results$sigma2Sample)  
sigma\_credInterval=quantile(bayes\_lin\_results$sigma2Sample, c(0.025, 0.975))  
rownames(beta\_credIntervals)=covNames  
beta\_credIntervals[which(rownames(beta\_credIntervals)=="rm"), ]  
  
## Interpretation: for one unit increase of rooms the hosing prices will rise between 3991,475 and 5009,826 dollars  
## with 95 % posterior probability.   
  
## b) Owner of house 381 is considering selling their house. Bought house for 10400  
  
old\_obs=as.vector(X[381,])  
new\_obs=old\_obs  
new\_obs[2]=10  
pred\_draw=rep(0,nIter)  
for (i in 1:nIter) {  
 pred\_draw[i]=bayes\_lin\_results$betaSample[i,]%\*%new\_obs+rnorm(1, mean=0,  
 sd=sqrt(bayes\_lin\_results$sigma2Sample[i]))  
}  
pred\_mean=mean(pred\_draw)  
hist(pred\_draw, breaks=50)  
quantile(posterior\_prices, c(0.025, 0.975))  
sum(pred\_draw>=30)/nIter  
  
## c) See paper.

##Assignment 4

## a) Simulate 1000 draws from the posterior distrib of theta using conjugate prior for theta with mean 250  
## and std = 50. Poisson likelihood.   
  
data=c(220,323,174,229)  
alpha=25  
beta=0.1  
n=length(data)  
  
logPriorGamma = function(theta, alpha, beta) {  
 return(dgamma(theta, 50, beta, log=TRUE))  
}  
  
logLike = function(data, theta) {  
 n=length(data)  
 first=sum(data)\*log(theta)  
 second=theta\*n  
 third=0  
 for (i in data) {  
 for (j in 1:i) {  
 third=third+log(j)  
 }  
 }  
 return(first-second-third)  
}  
  
logPosterior = function(data, theta, alpha, beta) {  
 prior=logPriorGamma(theta, alpha, beta)  
 likelihood=logLike(data, theta)  
 return(likelihood + prior)  
}  
  
# Conjugate prior for poisson is Gamma(alpha, beta), we know that posterior is Gamma(alpha + sum(data), beta+n)  
post\_draws=rgamma(1000, alpha+sum(data), beta+n)  
hist(post\_draws, main="Posterior distribution of theta", xlab=expression(theta))  
  
## b) Simulate 1000 draws from the predictive distrib of next quarter's demand, X5, and plot the draws  
## in histogram.   
  
q5=rpois(1000, post\_draws)  
hist(q5, breaks=50, main="Predictive distribution of quarter 5", xlab="Qty")  
sum(q5<=200)/1000  
  
## c)   
  
  
utility <- function(a,X5){  
 util = rep(0,length(X5))  
 util[X5<=a] = 10\*X5[X5<=a]-(a-X5[X5<=a])  
 util[X5>a] = 10\*a-0.05\*(X5[X5>a]-a)^2  
 return(util)  
}  
  
mean(q5)  
a=seq(136,336,1)  
results = matrix(0,length(q5),length(a))  
count=1  
nameVec=rep(0,length(a))  
for (i in a) {  
 results[,count]=utility(i,q5)  
 nameVec[count]=as.character(i)  
 count=count+1  
}  
opt\_vector=matrix(0,1,length(a))  
for (i in 1:length(a)) {  
 opt\_vector[i]=mean(results[,i])  
}  
colnames(opt\_vector)=nameVec  
opt\_decision=as.numeric(opt\_vector[,which(opt\_vector==max(opt\_vector))])  
names(opt\_vector[,which(opt\_vector==max(opt\_vector))])  
plot(a, opt\_vector, type="l", lwd=1, col="red")  
abline(v=as.numeric(names(opt\_vector[,which(opt\_vector==max(opt\_vector))])), col="blue")

#2017-10-27

##Assignment 1

## a) Likelihood: Beta symmetric, prior, expon(1). Plot posterior distrib.  
  
thetaGrid=seq(0.01, 15, length=1000)  
data=yProp  
lambda=1  
  
logPriorExp = function(theta, lambda) {  
 return(dexp(theta, rate=lambda, log=TRUE))  
}  
  
logPosterior = function(x, theta, lambda) {  
 prior=logPriorExp(theta, lambda)  
 likelihood=sum(dbeta(x, theta, theta, log=TRUE))  
 return(likelihood+prior)  
}  
  
theta\_post=sapply(thetaGrid, logPosterior, x=data, lambda=lambda)  
theta\_post\_norm=1/((15-0.01)/1000)\*exp(theta\_post)/sum(exp(theta\_post))  
plot(thetaGrid, theta\_post\_norm, type="l", lwd=2, xlab=expression(theta), ylab="Posterior density")  
  
# Zero to 1 loss means posterior mode is the optimal point estimator  
  
index=which(theta\_post\_norm==max(theta\_post\_norm))  
opt\_theta=thetaGrid[index]  
print(opt\_theta)  
  
## Optimal theta is around 4.481491  
  
## b) Theta1 and theta2 are independent apriori. Plot joint posterior distrib  
  
logPosteriorMult = function(theta, x, lambda) {  
 theta1=theta[1]  
 theta2=theta[2]  
 prior1=logPriorExp(theta1, lambda)  
 prior2=logPriorExp(theta2, lambda)  
 likelihood=sum(dbeta(x, theta1, theta2, log=TRUE))  
 return(likelihood+prior1+prior2)  
}  
  
# Defining initial values to be passed on to the optimizer  
initVal = c(1,1)  
  
# Finding the optimized betavector  
optimResult = optim(initVal, logPosteriorMult, x=data, lambda=1, method=c("L-BFGS-B"),  
 control=list(fnscale=-1), lower=c(0.01,0.01), upper=c(Inf, Inf), hessian=TRUE)  
  
# Defining the values of interest  
postMode = optimResult$par  
postCov = -solve(optimResult$hessian)  
names(postMode)=c("Theta1", "Theta2")  
rownames(postCov)=c("Theta1", "Theta2")  
colnames(postCov)=c("Theta1", "Theta2")  
print("The posterior mode is:")  
print(postMode)  
print("The approximated standard deviation is:")  
print(postCov)  
  
## c) Discuss how a Bayesian can determine if the symmetric model in 1a) or the non-symmetric model in 1b)   
## is most appropriate for this data. No need to compute anything here, just discuss.  
  
## By calculating marginal likelihood for each model and check which has the highest. One can also calculate  
## the bayes factor or the posterior model probabilities and choose the model with the highest probability.

##Assignment 2

## a) Use conjugate priors, standard normal and invchisq and use BayesLinReg to simulate 5000 draws from posterior  
## distrib  
  
# Reading the data from file  
library(MASS)  
BostonHousing = Boston  
y = BostonHousing$medv  
X = cbind(1,BostonHousing[,1:13]) # Adding a column of ones for the intercept  
names(X)[1] <- "intercept"  
covNames <- names(X)  
y <- as.numeric(y)  
X <- as.matrix(X)  
  
library(mvtnorm)  
  
# Defining a function that simulates from the scaled inverse Chi-square distribution  
rScaledInvChi2 <- function(n, df, scale){  
 return((df\*scale)/rchisq(n,df=df))  
}  
  
BayesLinReg <- function(y, X, mu\_0, Omega\_0, v\_0, sigma2\_0, nIter){  
 # Direct sampling from a Gaussian linear regression with conjugate prior:  
 #  
 # beta | sigma2 ~ N(mu\_0, sigma2\*inv(Omega\_0))  
 # sigma2 ~ Inv-Chi2(v\_0,sigma2\_0)  
 #   
 # Author: Mattias Villani, IDA, Linkoping University. http://mattiasvillani.com  
 #  
 # INPUTS:  
 # y - n-by-1 vector with response data observations  
 # X - n-by-nCovs matrix with covariates, first column should be ones if you want an intercept.  
 # mu\_0 - prior mean for beta  
 # Omega\_0 - prior precision matrix for beta  
 # v\_0 - degrees of freedom in the prior for sigma2  
 # sigma2\_0 - location ("best guess") in the prior for sigma2  
 # nIter - Number of samples from the posterior (iterations)  
 #  
 # OUTPUTS:  
 # results$betaSample - Posterior sample of beta. nIter-by-nCovs matrix  
 # results$sigma2Sample - Posterior sample of sigma2. nIter-by-1 vector  
   
 # Compute posterior hyperparameters  
 n = length(y) # Number of observations  
 nCovs = dim(X)[2] # Number of covariates  
 XX = t(X)%\*%X  
 betaHat <- solve(XX,t(X)%\*%y)  
 Omega\_n = XX + Omega\_0  
 mu\_n = solve(Omega\_n,XX%\*%betaHat+Omega\_0%\*%mu\_0)  
 v\_n = v\_0 + n  
 sigma2\_n = as.numeric((v\_0\*sigma2\_0 + ( t(y)%\*%y + t(mu\_0)%\*%Omega\_0%\*%mu\_0 - t(mu\_n)%\*%Omega\_n%\*%mu\_n))/v\_n)  
 invOmega\_n = solve(Omega\_n)  
   
 # The actual sampling  
 sigma2Sample = rep(NA, nIter)  
 betaSample = matrix(NA, nIter, nCovs)  
 for (i in 1:nIter){  
   
 # Simulate from p(sigma2 | y, X)  
 sigma2 = rScaledInvChi2(n=1, df = v\_n, scale = sigma2\_n)  
 sigma2Sample[i] = sigma2  
   
 # Simulate from p(beta | sigma2, y, X)  
 beta\_ = rmvnorm(n=1, mean = mu\_n, sigma = sigma2\*invOmega\_n)  
 betaSample[i,] = beta\_  
   
 }  
 return(results = list(sigma2Sample = sigma2Sample, betaSample=betaSample))  
}  
  
mu\_0=rep(0,ncol(X))  
omega\_0=1/100\*diag(ncol(X))  
v\_0=1  
sigma2\_0=36  
nIter=5000  
post\_distrib=BayesLinReg(y,X, mu\_0, omega\_0, v\_0, sigma2\_0, nIter)  
post\_beta=post\_distrib$betaSample  
colnames(post\_beta)=covNames  
lstat\_post=subset(post\_beta, select="lstat")  
par(mfrow=c(1,1))  
plot(density(lstat\_post), main="Posterior density of lstat", lwd=2)  
credInterval=quantile(lstat\_post, probs=c(0.05, 0.95))  
abline(v=credInterval[1], col="grey", lwd=3, lty=3)  
abline(v=credInterval[2], col="grey", lwd=3, lty=3)  
  
# Since posterior of beta is the student t-distrib the distrib is symmetric and therefore HPD interval is the same  
# as equal tail interval  
  
new\_obs=X[9,]  
names(new\_obs)=covNames  
new\_obs\_2=new\_obs  
new\_obs\_2[which(names(new\_obs)=="lstat")]=new\_obs\_2[which(names(new\_obs)=="lstat")]\*0.7  
post\_sigma2=post\_distrib$sigma2Sample  
pred\_price1=post\_beta%\*%new\_obs+rnorm(nIter, mean=0, sd=sqrt(post\_sigma2))  
pred\_price2=post\_beta%\*%new\_obs\_2+rnorm(nIter, mean=0, sd=sqrt(post\_sigma2))  
hist(pred\_price1, breaks=50, main="Histogram of predicted price before change")  
hist(pred\_price2, breaks=50, main="Histogran of predicted price after change")  
pred\_price\_house9=post\_beta[,14]\*(new\_obs[14]\*0.7-new\_obs[14])  
mean(pred\_price\_house9)  
quantile(pred\_price\_house9, probs=c(0.025, 0.975))  
  
# For a house like number 9 it will increase the house price with high posterior probability.

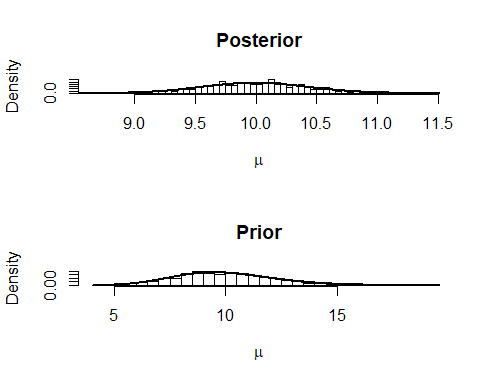
##Assignment 4

## a) Simulate 1000 draws from predictive distrib of the maximal weight on a given future day, model: y=10\*a where   
## y is the weight and a is the build cost. y~N(theta, sigma2). Noninformative prior assumed.  
  
y=c(191, 196, 197, 189)  
sigma2=10^2  
# Noninformative prior assumed to be constant  
  
yPred\_post=rnorm(1000, mean=mean(y), sd=sqrt(sigma2\*(1+1/length(y))))  
  
## b) Use simulation to approximate the predictive probability that weight higher than 230  
  
pred\_max365=rep(0,1000)  
for (i in 1:1000) {  
 pred\_max365[i]=max(rnorm(365, mean=mean(y), sd=sqrt(sigma2\*(1+1/length(y)))))  
}  
prob\_yPred365=sum(pred\_max365>230)/1000  
print(prob\_yPred365)  
  
## The probability is 0.157  
  
## c) The loss function is linear  
  
expectedLoss = function(a, maxWeight) {  
 probCollapse=sum(maxWeight>10\*a)/1000  
 return(a\*(1-probCollapse)+probCollapse\*(a+100))  
}  
  
a=seq(20,30,0.01)  
plot(a, sapply(a, expectedLoss, maxWeight=pred\_max365), type="l", lwd=2, xlab="a", ylab="EL",  
 main="Loss function")  
aOpt=a[which(min(sapply(a, expectedLoss, maxWeight=pred\_max365)))]  
print(aOpt)  
  
## The answer is 23.89

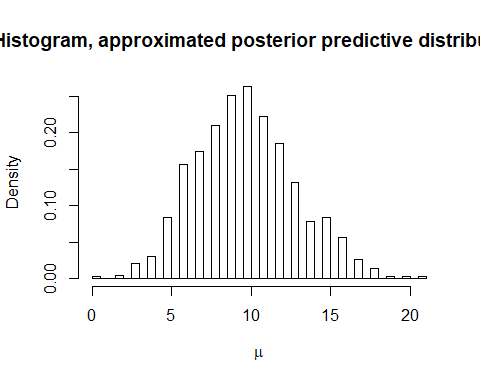
#2018-06-01

##Assignment 1

## a) Draw 1000 samples from prior (Gamma) and 1000 samples from posterior (Gamma). Plot prior and posterior using  
## both samples and their analytical expressions.   
  
n=50  
x\_mean=10  
beta=2  
nDraws=1000  
  
# We know that posterior distribution is the Gamma(alpha+sum(data), beta+n). Mean for Gamma distrib is alpha/beta.  
## If beta=2 then beta+n for posterior is 52. alpha/2=(alpha+500)/52 which yields 50\*alpha=1000 and alpha=20  
## Check: 20/2=10, (20+500)/52=10 OK!  
  
alpha=20 # According to motivation above  
post\_draws=rgamma(nDraws, alpha+n\*x\_mean, beta+n)  
prior\_draws=rgamma(nDraws, alpha, beta)  
gridWidth=0.01  
muGrid\_post=seq(7,12, gridWidth) # Range taken with inspiration from histogram  
muGrid\_prior=seq(4,20,gridWidth)  
par(mfrow=c(2,1))  
hist(post\_draws, breaks=50, main="Posterior", xlab=expression(mu),  
 freq=FALSE)  
lines(muGrid\_post, dgamma(muGrid\_post, alpha+n\*x\_mean, beta+n), lwd=2, xlab=expression(mu))  
hist(prior\_draws, breaks=50, main="Prior", xlab=expression(mu),  
 freq=FALSE)  
lines(muGrid\_prior, dgamma(muGrid\_prior, alpha, beta), lwd=2, xlab=expression(mu))



## As seen in the plots the distributions resemble each other.   
  
## b) Simulate 1000 draws from predictive distribution of new observation and plot distribution.  
  
par(mfrow=c(1,1))  
x\_pred=rpois(1000, lambda=post\_draws)  
hist(x\_pred, breaks=50, main="Histogram, approximated posterior predictive distribution", xlab=expression(mu),  
 freq=FALSE)



## c) Prob that x51=10 based on posterior predictive distribution  
  
sum(x\_pred==10)/nDraws

## [1] 0.132

##Assignemnt 2

## Linear regression model for fish with 3 covariates.   
  
# Reading the data from file  
load(file = 'fish.RData')  
  
library(mvtnorm)  
  
# Defining a function that simulates from the scaled inverse Chi-square distribution  
rScaledInvChi2 <- function(n, df, scale){  
 return((df\*scale)/rchisq(n,df=df))  
}  
  
BayesLinReg <- function(y, X, mu\_0, Omega\_0, v\_0, sigma2\_0, nIter){  
 # Direct sampling from a Gaussian linear regression with conjugate prior:  
 #  
 # beta | sigma2 ~ N(mu\_0, sigma2\*inv(Omega\_0))  
 # sigma2 ~ Inv-Chi2(v\_0,sigma2\_0)  
 #   
 # Author: Mattias Villani, IDA, Linkoping University. http://mattiasvillani.com  
 #  
 # INPUTS:  
 # y - n-by-1 vector with response data observations  
 # X - n-by-nCovs matrix with covariates, first column should be ones if you want an intercept.  
 # mu\_0 - prior mean for beta  
 # Omega\_0 - prior precision matrix for beta  
 # v\_0 - degrees of freedom in the prior for sigma2  
 # sigma2\_0 - location ("best guess") in the prior for sigma2  
 # nIter - Number of samples from the posterior (iterations)  
 #  
 # OUTPUTS:  
 # results$betaSample - Posterior sample of beta. nIter-by-nCovs matrix  
 # results$sigma2Sample - Posterior sample of sigma2. nIter-by-1 vector  
   
 # Compute posterior hyperparameters  
 n = length(y) # Number of observations  
 nCovs = dim(X)[2] # Number of covariates  
 XX = t(X)%\*%X  
 betaHat <- solve(XX,t(X)%\*%y)  
 Omega\_n = XX + Omega\_0  
 mu\_n = solve(Omega\_n,XX%\*%betaHat+Omega\_0%\*%mu\_0)  
 v\_n = v\_0 + n  
 sigma2\_n = as.numeric((v\_0\*sigma2\_0 + ( t(y)%\*%y + t(mu\_0)%\*%Omega\_0%\*%mu\_0 - t(mu\_n)%\*%Omega\_n%\*%mu\_n))/v\_n)  
 invOmega\_n = solve(Omega\_n)  
   
 # The actual sampling  
 sigma2Sample = rep(NA, nIter)  
 betaSample = matrix(NA, nIter, nCovs)  
 for (i in 1:nIter){  
   
 # Simulate from p(sigma2 | y, X)  
 sigma2 = rScaledInvChi2(n=1, df = v\_n, scale = sigma2\_n)  
 sigma2Sample[i] = sigma2  
   
 # Simulate from p(beta | sigma2, y, X)  
 beta\_ = rmvnorm(n=1, mean = mu\_n, sigma = sigma2\*invOmega\_n)  
 betaSample[i,] = beta\_  
   
 }  
 return(results = list(sigma2Sample = sigma2Sample, betaSample=betaSample))  
}  
  
## a) Plot marginal posterior for each param  
  
y=as.matrix(subset(fish, select="length"))  
X=as.matrix(fish[,2:ncol(fish)])  
covNames=colnames(X)  
mu\_0=rep(0,3)  
omega\_0=0.01\*diag(1,3)  
v\_0=1  
sigma2\_0=100^2  
nIter=5000  
  
linPost=BayesLinReg(y, X, mu\_0, omega\_0, v\_0, sigma2\_0, nIter)  
betaPost=linPost$betaSample  
colnames(betaPost)=covNames  
sigma2Post=linPost$sigma2Sample  
par(mfrow=c(2,2))  
for (i in 1:ncol(betaPost)) {  
 hist(betaPost[,i], xlab=paste("Beta",i,sep=""), main=paste("Marginal posterior distribution of beta", i, sep=""))  
}  
hist(sigma2Post, xlab=expression(sigma), main="Marginal posterior distribution of sigma2")  
  
par(mfrow=c(1,1))  
  
## Construct 90 % equal tail interval for beta1 and interpret it.  
  
quantile(subset(betaPost, select="age"), probs=c(0.05, 0.95))  
  
## It can be concluded that when the age of the fish increases with one unit the length of the fish increases   
## with approximately between 2.284 and 2.960 mm with 90 % posterior probability.   
  
## d) New experiment fish has been grown in water tank with water temp 30 degrees celsius. Newborn fish have  
## have been inserted into the tank at two time points, 30 days ago and 100 days ago. Equal amount of fish  
## in the two different ages. You pick up fish randomly from water tank. Do bayesian analysis (using sim methods)  
## to determine predictive distrib of the length of the picked up fish.   
  
x1=c(1,30,30)  
x2=c(1,100,30)  
x\_pred=rep(0,nIter)  
for (i in 1:nIter) {  
 prob=runif(1)  
 if(prob>0.5) {  
 x\_pred[i]=betaPost[i,]%\*%x1+rnorm(1, mean=0, sd=sqrt(sigmaPost[i]))  
 } else {  
 x\_pred[i]=betaPost[i,]%\*%x2+rnorm(1, mean=0, sd=sqrt(sigmaPost[i]))  
 }  
}  
hist(x\_pred, main="Histogram of predictive distribution of length of fish",  
 xlab="Length in mm", freq=FALSE, breaks=50)

##Assignment 3

## c) Choose between three models where two of them use Beta prior and the last one assumes p=0.5. Which model   
## should be chosen?  
  
model1=choose(10,3)\*gamma(4)\*gamma(8)\*gamma(2)/gamma(12)  
model2=choose(10,3)\*gamma(7)\*gamma(11)\*gamma(8)/(gamma(4)\*gamma(4)\*gamma(18))  
model3=choose(10,3)\*0.5^10  
model1\_norm=model1/sum(c(model1, model2, model3))  
model2\_norm=model2/sum(c(model1, model2, model3))  
model3\_norm=model3/sum(c(model1, model2, model3))

##Assignment 4

## a) Consider observations with values above 200. Remaining datapoints assumed to be indep. and follow a   
## truncated normal distribution with density specified. L=200 lower truncation point. Write a function in R  
## that computes the (unnormalized) log posterior distribution of mu. Use function to plot the posterior distrib  
## of mu for the observations greater than 200 in the data vector sulfur. For the plot, use a grid constructed  
## in R with seq(100,400,1)  
  
# Reading the data from file  
load(file = 'sulfur.RData')  
  
muGrid=seq(100,400,1)  
sigma=100  
data=sulfur[sulfur>200]  
  
# Constant prior for mu is assumed  
  
logPost = function(data, mu, sigma, L=200) {  
 nominator=dnorm((data-mu)/sigma, mean=0, sd=1, log=TRUE)  
 denominator=log(sigma)+log(1-pnorm((L-mu)/sigma))  
 return(sum(nominator-denominator+0)) # Assumed constant prior which can be set to 1 which in log scale is 0  
}  
  
post\_mu=exp(sapply(muGrid, logPost, data=data, sigma=sigma))  
post\_mu\_norm=post\_mu/sum(post\_mu) # Since gridwidth is 1 we don't have to compensate for it  
plot(muGrid, post\_mu\_norm, type="l", lwd=2, main="Posterior distribution of mu", xlab=expression(mu))  
  
library(rstan)  
T = length(sulfur)  
T\_cens = sum(sulfur <= 200)  
censData <- list(T=T, T\_cens = T\_cens, x=sulfur, L=200)  
  
# Model  
censModel <- '  
data {  
 int<lower=0> T; // Total number of time points  
 int<lower=0> T\_cens; // Number of censored time points  
 real x[T]; // Partly censored data  
 real<upper=max(x)> L; // Lower truncation point  
}  
  
parameters {  
 real mu;  
 real<lower=0> sigma;  
 real<upper=L> x\_cens[T\_cens]; // Censored values  
}  
  
model {  
 int t\_cens = 0;  
 for (t in 1:T){  
 if (x[t] > L)   
 x[t] ~ normal(mu,sigma);  
 else {  
 t\_cens += 1;  
 x\_cens[t\_cens] ~ normal(mu,sigma);  
 }  
 }  
}  
'  
  
## b) Now condiser all data points. Values below 200 being censored.   
  
fit=stan(model\_code=censModel, data=censData)  
print(fit)  
post\_draws=extract(fit)  
grid=seq(1,4000,1)  
plot(grid, post\_draws$mu, type="l", main="Traceplot of mu", xlab=expression(mu), ylab="Value")  
plot(grid, post\_draws$sigma, type="l", main="Traceplot of sigma", xlab=expression(sigma), ylab="Value")  
par(mfrow=c(4,2))  
for (i in 1:8) {  
 plot(grid, post\_draws$x\_cens[,i], type="l", main=paste("Traceplot of ", i, "th obs of obs below 200", sep=""),  
 xlab=i, ylab="Value")   
}  
par(mfrow=c(1,1))  
  
plot(post\_draws$mu, post\_draws$sigma, type="p", col="grey", main="Joint posterior of mu and sigma",  
 xlab=expression(mu), ylab=expression(sigma))  
  
## c) Instead consider time series model. Assume that observations follow an independent normal distrib  
## when conditioned on a latent AR(1) process z, but with values of xi below 200 being censored and set to 200.  
## Modify the stan code in order to do inference for this model instead. Also put a normal prior on   
## mu~N(300,100^2) Plot the posterior of phi. Also produce a plot that contains both the data and the posterior   
## mean and 95 % credible intervals for the latent intensity z over time.   
  
StanModel\_AR = '  
data {  
 int<lower=0> T; // Total number of time points  
 int<lower=0> T\_cens; // Number of censored time points  
 real x[T]; // Partly censored data  
 real<upper=max(x)> L; // Lower truncation point  
}  
  
parameters {  
 real mu;  
 real phi;  
 real<lower=0> sigma;  
 real<upper=L> x\_cens[T\_cens]; // Censored values  
 vector[T] z;  
}  
  
model {  
 // Prior  
 int t\_cens = 0;  
 phi ~ uniform(-1,1);  
 mu ~ normal(300, 100);  
 for (n in 2:T)  
 z[n] ~ normal(mu + phi \* (z[n-1]-mu), sigma);  
   
  
 // Model/likelihood  
 for (t in 1:T){  
 if (x[t] > L)   
 x[t] ~ normal(z[t],20);  
 else {  
 t\_cens += 1;  
 x\_cens[t\_cens] ~ normal(z[t],20);  
 }  
 }  
}  
  
generated quantities {  
 vector[T] post\_mean;  
 post\_mean = z;  
}  
'  
fitAR=stan(model\_code=StanModel\_AR, data=censData)  
print(fitAR)  
post\_draws\_AR=extract(fitAR)  
postPhi=post\_draws\_AR$phi  
postZ=post\_draws\_AR$post\_mean  
hist(postPhi, breaks=50, main="Approximated posterior density of phi", xlab=expression(phi), freq=FALSE)  
grid=seq(1,31)  
plot(grid, sulfur, col="blue", main="Emissions of sulfur dioxide", xlab="Day of month", ylab="mg/Nm^3",  
 ylim=c(0,500))  
postMean=rep(0,ncol(postZ))  
credIntervals=matrix(0,ncol(postZ),2)  
for (i in 1:ncol(postZ)) {  
 postMean[i]=mean(postZ[,i])  
 credIntervals[i,]=quantile(postZ[,i], probs=c(0.025, 0.975))  
}  
lines(grid, postMean, type="l", col="red", lwd=2)  
lines(grid, credIntervals[,1], col="grey", lwd=1, lty=2)  
lines(grid, credIntervals[,2], col="grey", lwd=1, lty=2)  
legend("topleft", legend=c("Data", "Posterior mean", "95 % cred intervals"), lwd=c(NaN, 2, 1), lty=c(NaN,1,2),  
 pch=c(1,NaN, NaN), col=c("blue", "red", "grey"))

#2019-08-21

##Assignment 1

## a) Use BayesLinReg to sim 5000 draws from posterior distrib of all coeff coefficients. Summarize posterior  
## with point estimate under quadratic loss function and 95 % equal tail intervals. Interpret cred intervals for  
## regression coefficient on nitrogen oxides concentration.  
  
###############################  
########## Problem 1 ##########   
###############################   
  
# Reading the data from file  
library(MASS)  
BostonHousing = Boston  
y = BostonHousing$medv  
X = cbind(1,BostonHousing[,1:13]) # Adding a column of ones for the intercept  
names(X)[1] <- "intercept"  
covNames <- names(X)  
y <- as.numeric(y)  
X <- as.matrix(X)  
XNewHouse <- c(1,0.03,40,1.5,0,0.5,6,30,5,3,300,17,390,4)  
  
if(length((grep("mvtnorm",installed.packages()[,1])))==0)  
 install.packages("mvtnorm")  
library(mvtnorm)  
  
# Defining a function that simulates from the scaled inverse Chi-square distribution  
rScaledInvChi2 <- function(n, df, scale){  
 return((df\*scale)/rchisq(n,df=df))  
}  
  
BayesLinReg <- function(y, X, mu\_0, Omega\_0, v\_0, sigma2\_0, nIter){  
 # Direct sampling from a Gaussian linear regression with conjugate prior:  
 #  
 # beta | sigma2 ~ N(mu\_0, sigma2\*inv(Omega\_0))  
 # sigma2 ~ Inv-Chi2(v\_0,sigma2\_0)  
 #   
 # Author: Mattias Villani, IDA, Linkoping University. http://mattiasvillani.com  
 #  
 # INPUTS:  
 # y - n-by-1 vector with response data observations  
 # X - n-by-nCovs matrix with covariates, first column should be ones if you want an intercept.  
 # mu\_0 - prior mean for beta  
 # Omega\_0 - prior precision matrix for beta  
 # v\_0 - degrees of freedom in the prior for sigma2  
 # sigma2\_0 - location ("best guess") in the prior for sigma2  
 # nIter - Number of samples from the posterior (iterations)  
 #  
 # OUTPUTS:  
 # results$betaSample - Posterior sample of beta. nIter-by-nCovs matrix  
 # results$sigma2Sample - Posterior sample of sigma2. nIter-by-1 vector  
   
 # Compute posterior hyperparameters  
 n = length(y) # Number of observations  
 nCovs = dim(X)[2] # Number of covariates  
 XX = t(X)%\*%X  
 betaHat <- solve(XX,t(X)%\*%y)  
 Omega\_n = XX + Omega\_0  
 mu\_n = solve(Omega\_n,XX%\*%betaHat+Omega\_0%\*%mu\_0)  
 v\_n = v\_0 + n  
 sigma2\_n = as.numeric((v\_0\*sigma2\_0 + ( t(y)%\*%y + t(mu\_0)%\*%Omega\_0%\*%mu\_0 - t(mu\_n)%\*%Omega\_n%\*%mu\_n))/v\_n)  
 invOmega\_n = solve(Omega\_n)  
   
 # The actual sampling  
 sigma2Sample = rep(NA, nIter)  
 betaSample = matrix(NA, nIter, nCovs)  
 for (i in 1:nIter){  
   
 # Simulate from p(sigma2 | y, X)  
 sigma2 = rScaledInvChi2(n=1, df = v\_n, scale = sigma2\_n)  
 sigma2Sample[i] = sigma2  
   
 # Simulate from p(beta | sigma2, y, X)  
 beta\_ = rmvnorm(n=1, mean = mu\_n, sigma = sigma2\*invOmega\_n)  
 betaSample[i,] = beta\_  
   
 }  
 return(results = list(sigma2Sample = sigma2Sample, betaSample=betaSample))  
}  
  
mu\_0=rep(0, ncol(X))  
omega\_0=1/10^2\*diag(ncol(X))  
v\_0=1  
sigma2\_0=5^2  
nIter=5000  
linPost=BayesLinReg(y, X, mu\_0, omega\_0, v\_0, sigma2\_0, nIter)  
betaPost=linPost$betaSample  
sigma2Post=linPost$sigma2Sample  
results=matrix(0,ncol(X)+1,3)  
results\_names=covNames  
results\_names=append(results\_names, "sigma2")  
rownames(results)=results\_names  
colnames(results)=c("Point estimator", "2,5%", "97,5%")  
for (i in 1:ncol(X)) {  
 results[i,1]=mean(betaPost[,i])  
 results[i,-1]=quantile(betaPost[,i], probs=c(0.025, 0.975))  
}  
results[(ncol(X)+1),1]=mean(sigma2Post)  
results[(ncol(X)+1),-1]=quantile(sigma2Post, probs=c(0.025, 0.975))  
results  
  
## b) Kernel density estimates. Compute posterior mode and HPD 90 % for sigma2  
  
sigma2\_kernel=density(sigma2Post)  
sigma2\_kernel.df=data.frame(sigma2=sigma2\_kernel$x, density=sigma2\_kernel$y)  
sigma2\_kernel.df=sigma2\_kernel.df[order(-sigma2\_kernel.df[,2]),]  
index=dim(sigma2\_kernel.df)[1]  
sigma2\_kernel.df$density=cumsum(sigma2\_kernel.df$density)/sum(sigma2\_kernel.df$density)  
sigma2Cred=sigma2\_kernel.df[sigma2\_kernel.df$density<0.9,]  
credInterval=c(min(sigma2Cred$sigma2), max(sigma2Cred$sigma2))  
sigma2Mode=sigma2\_kernel.df[1,]$sigma2  
  
plot(sigma2\_kernel, type="l", lwd=2, main="Kernel density estimate of sigma2", xlab=expression(sigma^2))  
abline(v=sigma2Mode, col="red", lwd=1, lty=2)  
abline(v=credInterval[1], col="grey", lwd=1, lty=3)  
abline(v=credInterval[2], col="grey", lwd=1, lty=3)  
legend("topright", legend=c("Kernel density estimate", "Posterior mode", "90 % HPD Interval"), lty=c(1,2,3),  
 lwd=c(2,1,1), col=c("black", "red", "grey"))  
  
## c) Construction company planning to build a new house with covariates given in XNewHouse. Cost is 20000 dollars  
## and the company is planning to sell the house when finished. Do Bayesian analysis to determine how probable  
## it is that the company will make money (that the house will sell for more than 20000 dollars).  
  
XNewHouse <- c(1,0.03,40,1.5,0,0.5,6,30,5,3,300,17,390,4)  
profitVec=rep(0,nIter)  
for (i in 1:nIter) {  
 profitVec[i]=-20+betaPost[i]%\*%XNewHouse+rnorm(1, mean=0, sd=sqrt(sigma2Post[i]))  
}  
hist(profitVec)  
probProfit=sum(profitVec>0)/nIter  
print(probProfit)  
quantile(profitVec, probs=c(0.025, 0.975))  
  
## Very probable that the company will make a profit since 98.82 % of the posterior draws are above zero. Negative  
## values are also not present in the 95 % equal tail interval which also indicates that the company will make  
## a profit.

##Assignment 2

## b) Simulate predictive draw of max no. of years until next earthquake occurs, 95 % prob. alpha=1, beta=1.   
  
alpha=1  
beta=1  
xObs=c(35, 14, 4, 10, 2)  
n=length(xObs)i  
nIter=5000  
predDistrib=rep(0,nIter)  
for(i in 1:nIter) {  
 posteriorDraw=rbeta(1,alpha+n, beta+sum(xObs))  
 predDistrib[i]=rgeom(1,posteriorDraw)  
}  
predDistrib\_maxYear=quantile(predDistrib, probs=0.95)  
predDistrib\_maxYear

##Assignment 3

## c) Calc unnormalized posterior and plot normalized posterior. Gamma prior and indep likelihoods.  
  
gridWidth=0.01  
thetaGrid=seq(0,2,gridWidth)  
xData <- c(1.888, 2.954, 0.364, 0.349, 1.090, 7.237)  
yData <- c(-1.246, -1.139, -0.358, -1.308, -0.930, -0.157, -0.111, -0.635)  
alpha=3  
beta=2  
  
logPosteriorX = function(theta, alpha, beta) {  
 return(dgamma(theta, alpha, beta, log=TRUE))  
}  
  
likeY = function(y, theta) {  
 return(-3\*sum(log(1+(1/5)\*(y-log(theta))^2)))  
}  
  
logPosterior = function(theta, alpha, beta, xDat, yDat) {  
 likelihoodY=likeY(yDat, theta)  
 logPostX=logPosteriorX(theta, length(xDat+3), sum(xDat)+2)  
 return(likelihoodY+logPostX)  
}  
  
post\_theta=sapply(thetaGrid, logPosterior, alpha=alpha, beta=beta, xDat=xData, yDat=yData)  
post\_theta\_norm=1/gridWidth\*exp(post\_theta)/sum(exp(post\_theta))  
plot(thetaGrid, post\_theta\_norm, type="l", lwd=2, main="Posterior of theta", xlab=expression(theta),  
 ylab="Density")

##Assignment 4

## Aircraft incidents assumed to be independent, follow negative binomial distrib. Assume joint prior   
## 1/phi^2  
## a) Simulate from posterior using Metropolis algorithm. Denote theta=c(mu, phi) and use as proposal dens  
## the multivariate normal density (random walk metropolis).  
  
# Load airline incidents data  
load(file = 'incidents.RData')  
data=incidents$incidents  
library(mvtnorm)  
  
nIter=1000  
burnIn=50  
theta\_0=c(200,20)  
c=0.1  
postCov=diag(c(100,5))  
  
# Defining function for sampling through metropolishastings  
RVMSampler = function(previousVal, postCov, c, myFunction, ...) {  
 proposalVal=rmvnorm(1, mean=previousVal, sigma=c\*postCov)  
 proposalVal[proposalVal<=0]=1e-6  
 alpha=min(1, exp(myFunction(proposalVal,...)-myFunction(previousVal, ...)))  
 u=runif(1)  
 if(u < alpha) {  
 return(list(theta=proposalVal, acceptProb=alpha))  
 } else {  
 return(list(theta=previousVal, acceptProb=alpha))  
 }  
}  
  
logPrior = function(phi) {  
 return(-2\*log(phi))  
}  
  
logLike <- function(param, x){  
 theta1 = param[1]  
 theta2 = param[2]  
 logPost = sum(logdNegBin(x, theta1, theta2)) - 2\*log(theta2)  
 return(logPost)  
}  
  
logPost = function(theta, data) {  
 log\_Prior=logPrior(theta[2])  
 log\_Like=logLike(theta, data)  
 return(log\_Prior+log\_Like)  
}  
  
post\_matrix = matrix(0, nIter+burnIn, 2)  
# Setting initial values of beta to same initVals as in the optimizer (taken randomly from normal distrib)  
post\_matrix[1,]=theta\_0  
accProb=rep(0, nIter)  
set.seed(12345)  
  
for(i in 1:(nIter+burnIn)) {  
 if(i<(nIter+burnIn)) {  
 draw=RVMSampler(post\_matrix[i,], postCov, c, logPost, data)  
 post\_matrix[i+1,]=draw$theta  
 accProb[i+1]=draw$acceptProb  
 }  
}  
  
iter=seq(1,nIter+burnIn,1)  
plot(iter[-(1:burnIn)], post\_matrix[-(1:burnIn),1], type="l", lwd=1, col="grey", main="Traceplot of mu in RVM",  
 xlab=expression(mu), ylab="Value")  
plot(iter[-(1:burnIn)], post\_matrix[-(1:burnIn),2], type="l", lwd=1, col="grey", main="Traceplot of phi in RVM",  
 xlab=expression(phi), ylab="Value")  
mean(accProb)  
  
## This MCMC sampler is not efficient since it moves very slowly and is therefore probably not exploring  
## the whole posterior distribution.We can also see that the acceptance probability for this algorithm  
## is around 84,4 % and it should be around 30 %. Once could tune the c param to lower the acceptance probability.  
## One example is to increase c to a value of 3 which would yield in approximately 30 % acceptance rate.   
  
## b) Instead simulate from posterior using metropolis hastings.   
  
c=0.8  
  
MHSampler = function(previousVal, postCov, c, myFunction, ...) {  
 proposalVal\_mu=rgamma(1, c\*previousVal[1], c)  
 proposalVal\_phi=rgamma(1, c\*previousVal[2], c)  
 proposalVal=c(proposalVal\_mu, proposalVal\_phi)  
 proposalVal[proposalVal<=0]=1e-6  
 alpha=min(1, exp(myFunction(proposalVal,...)-myFunction(previousVal, ...)+  
 dgamma(previousVal[1], c\*proposalVal[1], c)+dgamma(previousVal[2],c\*proposalVal[2],c)-  
 dgamma(proposalVal[1], c\*previousVal[1], c)-dgamma(proposalVal[2], c\*proposalVal[2],c)))  
 u=runif(1)  
 if(u < alpha) {  
 return(list(theta=proposalVal, acceptProb=alpha))  
 } else {  
 return(list(theta=previousVal, acceptProb=alpha))  
 }  
}  
  
post\_matrix2 = matrix(0, nIter+burnIn, 2)  
theta\_0=c(200,10)  
post\_matrix2[1,]=theta\_0  
accProb2=rep(0, nIter)  
set.seed(12345)  
  
for(i in 1:(nIter+burnIn)) {  
 if(i<(nIter+burnIn)) {  
 draw=MHSampler(post\_matrix2[i,], postCov, c, logPost, data)  
 post\_matrix2[i+1,]=draw$theta  
 accProb2[i+1]=draw$acceptProb  
 }  
}  
  
plot(iter[-(1:burnIn)], post\_matrix2[-(1:burnIn),1], type="l", lwd=1, col="grey", main="Traceplot of mu in MH",  
 xlab=expression(mu), ylab="Value")  
plot(iter[-(1:burnIn)], post\_matrix2[-(1:burnIn),2], type="l", lwd=1, col="grey", main="Traceplot of phi in MH",  
 xlab=expression(phi), ylab="Value")  
mean(accProb2)  
  
## The new algorithm seems to rapidly explore the posterior which is good. The acceptance probability is also lower  
## around 30 % which also indicates that this algorithm is better than the previous one.

#2019-10-31

##Assignment 1

## a) Theta is known  
  
theta=0.6  
eu\_buy=0.6\*30-0.4\*10  
eu\_nobuy=0.6\*90-120\*0.4  
  
## Answer: Should buy option  
  
## c) Compute bayesian decision for day 101 based on information in b).   
  
theta\_new=13/21  
eu\_buy\_new=theta\_new\*30-(1-theta\_new)\*10  
eu\_nobuy\_new=theta\_new\*90-(1-theta\_new)\*120  
  
## Answer: Should buy since utility higher.

##Assignment 2

## a) Consider poisson likelihood model. Use conjugate prior and plot posterior in given interval.   
## Compute posterior probability that theta is smaller than 21.   
  
# Calculations show that alpha=20, beta=1  
  
data=Traffic$y  
alpha=20  
beta=1  
n=length(data)  
  
# We know that Poisson with gamma prior is gamma distributed with alphaNew=alpha+sum(data), betaNew=beta+n  
  
grid=seq(18,24,0.01)  
post\_distrib=dgamma(grid, shape=alpha+sum(data), rate=beta+n)  
plot(grid, post\_distrib, type="l", lwd=2, main="Posterior distrib. of theta", xlab=expression(theta))  
post\_prob=pgamma(21, shape=alpha+sum(data), rate=beta+n)  
  
## Answer: Probability is 0.0557  
  
## b) Two independent poisson models.   
  
data\_model1=Traffic[which(Traffic[,3]=="yes"),]$y  
data\_model2=Traffic[which(Traffic[,3]=="no"),]$y  
  
alpha\_1=20+sum(data\_model1)  
alpha\_2=20+sum(data\_model2)  
beta\_1=1+length(data\_model1)  
beta\_2=1+length(data\_model2)  
post\_distrib\_1=rgamma(5000, shape=alpha\_1, rate=beta\_1)  
post\_distrib\_2=rgamma(5000, shape=alpha\_2, rate=beta\_2)  
hist(post\_distrib\_1, breaks=50)  
hist(post\_distrib\_2, breaks=50)  
post\_diff=post\_distrib\_2-post\_distrib\_1  
hist(post\_diff,  
 main="Posterior distribution of difference between no speedlimit and speedlimit", xlab="No. of accidents")  
quantile(post\_diff, prob=c(0.025, 0.975))  
mean(post\_diff)  
  
## We can see that the difference between the two distributions is larger than 0 with high probability. In this  
## case we can say that the difference in traffic accidents between when no speed limit were applied and   
## when a speed limit were applied is between 2.82 and 5.53 approximately with 95 % posterior probability.   
## The conclusion from this is that yes, a speed limit leads to a lower amount of accidents.  
  
## c) A politician claims that the experiment proves that introducing speed limit decreases the number  
## of accidents by at least 15 %.   
  
mean(0.85\*post\_distrib\_2>post\_distrib\_1)  
  
## Likely that the decrease yields 15 % but 86 % probable and not 95 % probability which is commonly used  
## in statistical experiments.

##Assignment 3

## c) Make simulations of joins posterior of v and pi using Gibbs sampling.  
  
x=20  
lambda=10  
alpha=2  
beta=2  
nIter=2000  
burnIn=500  
  
results=matrix(0,burnIn+nIter,2)  
initVal=lambda # Since lambda=30  
results[1,1]=initVal  
results[1,2]=rnorm(1)  
for (i in 1:(nIter+burnIn-1)) {  
 z=rpois(1, lambda\*(1-results[i,2]))  
 results[i+1,1]=z+x  
 results[i+1,2]=rbeta(1, alpha+x, beta+results[i+1,1]-x)  
}  
  
grid=seq(burnIn+1, nIter+burnIn)  
barplot(table(results[(burnIn+1):(nIter+burnIn),1]), main="Marginal posterior of nu", xlab=expression(nu))  
hist(results[(burnIn+1):(nIter+burnIn),2], breaks=50, main="Marginal posterior of pi", xlab=expression(pi))  
plot(grid, results[(burnIn+1):(nIter+burnIn),2],type="l")  
plot(grid, results[(burnIn+1):(nIter+burnIn),1], type="l")  
  
## Convergence seems good since markov chain is exploring full posterior and have good mixing.

##Assignment 4

## a) Use supplied stan model to do Bayesian inference. Draw 2000 posterior samples and use 500 for burnin.   
## Produce figure with scatter plot, overlay curve for mean of posterior predictive distrib, in range [0,25].   
## Also overlay curves 90 % equal tail interval for same posterior predictive distrib given values of x in range [0,25]  
  
# Load data  
cars = cars  
  
library(rstan)  
LinRegModel <- '  
data {  
 int<lower=0> N;  
 vector[N] x;  
 vector[N] y;  
}  
parameters {  
 real alpha;  
 real beta;  
 real<lower=0> sigma2;  
}  
model {  
 sigma2 ~ scaled\_inv\_chi\_square(5,10);  
 for (n in 1:N)  
 y[n] ~ normal(alpha + beta \* x[n], sqrt(sigma2));  
}  
'  
x=cars$speed  
y=cars$dist  
nIter=2000  
burnIn=500  
N=dim(cars)[1]  
data=list(N=N,x=x,y=y)  
fit=stan(model\_code=LinRegModel, data=data, iter=nIter, warmup = 500, chains=1)  
print(fit)  
postDraws=extract(fit)  
alpha\_draws=postDraws$alpha  
beta\_draws=postDraws$beta  
sigma\_draws=postDraws$sigma2  
xGrid=seq(0,25)  
n=length(alpha\_draws)  
mean\_credInt=matrix(0,length(xGrid),3)  
count=1  
for (i in 1:length(xGrid)) {  
 ysim=rep(0,length(nIter-burnIn))  
 ysim=alpha\_draws+beta\_draws\*xGrid[i]+rnorm(nIter-burnin, mean=0, sd=sqrt(sigma\_draws))  
 mean\_credInt[count,1]=mean(ysim)  
 mean\_credInt[count,-1]=quantile(ysim, probs = c(0.05, 0.95))  
 count=count+1  
}  
  
plot(x,y,xlab="Speed", ylab="Distance", col="blue", main="Plot for model with constant sigma prior")  
lines(xGrid, mean\_credInt[,1], lwd=2, col="red")  
lines(xGrid, mean\_credInt[,2], lwd=1, lty=2)  
lines(xGrid, mean\_credInt[,3], lwd=1, lty=2)  
legend("topleft", legend=c("Data", "Posterior mean", "90 % cred interval"), col=c("blue", "red", "grey"),   
 pch=c(1, NaN, NaN), lty=c(NaN, 1, 2), lwd=c(NaN, 2, 1))  
  
## b) Compute 95 % equal tail credible interval for alpha. Give real-world interpret of the interval.   
  
quantile(alpha\_draws, probs=c(0.025, 0.975))  
  
## The interpretation of the credible interval for alpha is that if the car has no speed it travels a negative  
## distance between -31 and 4.25 approximately with 95 % posterior probability. This is not realistic. To prevent this  
## a prior can be set to alpha with a mean around zero which however would make the linear prediction worse.   
## One can also use the log Normal distribution for y to force it to have a value above zero.   
  
## c) Reproduce results in b) with heteroscadastic variance.   
  
LinRegModel\_hetero <- '  
data {  
 int<lower=0> N;  
 vector[N] x;  
 vector[N] y;  
}  
parameters {  
 real alpha;  
 real beta;  
 real<lower=0> sigma2[N];  
 real gamma;  
 real phi;  
}  
model {  
 for (n in 1:N)  
 sigma2[n] ~ scaled\_inv\_chi\_square(5,exp(gamma+phi\*x[n]));  
 for (n in 1:N)  
 y[n] ~ normal(alpha + beta \* x[n], sqrt(sigma2[n]));  
}  
'  
data=list(N=N,x=x,y=y)  
fit2=stan(model\_code=LinRegModel\_hetero, data=data, iter=nIter, warmup = 500, chains=1)  
print(fit2)  
postDraws2=extract(fit2)  
alpha\_draws=postDraws2$alpha  
beta\_draws=postDraws2$beta  
sigma\_draws=postDraws2$sigma2  
xGrid=seq(0,25)  
n=length(alpha\_draws)  
mean\_credInt=matrix(0,length(xGrid),3)  
count=1  
for (i in 1:length(xGrid)) {  
 rinv=rchisq(nIter-burnIn, 5)  
 sigma\_draw=5\*exp(postDraws2$gamma + xgrid[i] \* postDraws2$phi)^2/rinv  
 ysim=rep(0,length(nIter-burnIn))  
 ysim=alpha\_draws+beta\_draws\*xGrid[i]+rnorm(nIter-burnin, mean=0, sd=sqrt(sigma\_draw))  
 mean\_credInt[count,1]=mean(ysim)  
 mean\_credInt[count,-1]=quantile(ysim, probs = c(0.05, 0.95))  
 count=count+1  
}  
  
plot(x,y,xlab="Speed", ylab="Distance", col="blue", main="Plot of model with heteroscadastic sigma prior")  
lines(xGrid, mean\_credInt[,1], lwd=2, col="red")  
lines(xGrid, mean\_credInt[,2], lwd=1, lty=2)  
lines(xGrid, mean\_credInt[,3], lwd=1, lty=2)  
legend("topleft", legend=c("Data", "Posterior mean", "90 % cred interval"), col=c("blue", "red", "grey"),   
 pch=c(1, NaN, NaN), lty=c(NaN, 1, 2), lwd=c(NaN, 2, 1))  
  
## The new model seems to capture the data better than the old one.