Lab3 Report

aswma317,varsi146

2023-05-13

```r
####Question 1: Gibbs sampler for Normal model####

#Get data
data <- readRDS('Precipitation.rds')
#ln(data) is Normally distributed
ln_data <- log(data)
n <- length(data)

#Prior mu is N(mu0, tau0_sq)
#Prior sigma_sq is N(nu0, sigma0_sq)
#Initializing the prior parameters
pr_mu0 <- 1; pr_tau0_sq <- 1; pr_nu0 <- 1; pr_sigma0_sq <- 1

#Question a.part 1: simulate from joint posterior from full conditional L7,S16

#Gibbs sampler - only when the full conditional distr is known
#Initial values for posterior pos_sigma_sq:
#set it from fitdistr(ln_data,'normal'), can check if pos_mu matches the output
#pos_sigma_sq <- 1.32087052
pos_mu <- 1.30820257
nDraws <- 1000
gibbsDraws <- matrix(0,nDraws,2)#1-Mean, 2-Var

for (i in 1:nDraws) {
  #Get the nu_n and pos_sigma_sq
  nu_n <- pr_nu0 + n
  #L3,S5 - Draw from scaled inverse chi-square same as in Lab2
  X <- rchisq(n = 1, df = nu_n)
  pos_sigma_sq <- (nu_n *
                     ((pr_nu0*pr_sigma0_sq) + sum((ln_data-pos_mu)^2))/(nu_n))/X
  # Alternative Implementation for posterior of sigma as per L7 i.e, full
  # conditional posterior of sigma in Gibbs Sampling.
  # pos_sigma_sq <- rinvchisq(n = 1, df = nu_n,
  #                           scale = ((pr_nu0*pr_sigma0_sq) +
  #                           sum((ln_data-pos_mu)^2))/(nu_n))
  gibbsDraws[i,2] <- pos_sigma_sq

  #Get the mu_n(mean) and tau_n_sq(var) for pos_mu which is N(mu_n, tau_n_sq)
  # Part of Full Conditional posterior for mean in Gibbs Sampling
  # As part of derivation for Normal Model - Known variance - Normal Prior
  mu_n <- mean(ln_data) * (pr_tau0_sq/(pr_tau0_sq + (pos_sigma_sq/n))) +
    pr_mu0 * (pos_sigma_sq/((n*pr_tau0_sq) + pos_sigma_sq))
  tau_n_sq <- (pr_tau0_sq * pos_sigma_sq)/
    ((n*pr_tau0_sq) + pos_sigma_sq)
  # Full conditional Posterior - L7
  pos_mu <- rnorm(n = 1, mean = mu_n, sd = sqrt(tau_n_sq))
  gibbsDraws[i,1] <- pos_mu
}
gibbs_posterior <- apply(gibbsDraws, 2, mean)

print('The mean of the fully conditional posterior parameters are:\n')
print(gibbs_posterior)

#Question a.part 2: Evaluate convergence by calculating the Inefficiency Factor
#and by plotting the trajectories of the sampled Markov chains.

# acf calculates the auto-correlation between draws for given lag
a_mu_Gibbs <- acf(gibbsDraws[,1])
a_var_Gibbs <- acf(gibbsDraws[,2])

IF_mu_Gibbs <- 1+2*sum(a_mu_Gibbs$acf[-1])
IF_var_Gibbs <- 1+2*sum(a_var_Gibbs$acf[-1])

print(paste('The inefficiency factor of Gibbs draws with respect to posterior mean is:',
            round(IF_mu_Gibbs,2)))
print(paste('The inefficiency factor of Gibbs draws with respect to posterior sigma-squared is:',
            round(IF_var_Gibbs,2)))


# Traceplot of Gibbs draws
# par(mfrow=c(2,1))
plot(1:nDraws, gibbsDraws[,1], type = "l",#col="red",
     xlab = 'Iterations', ylab = 'Posterior Mean')
abline(h = mean(ln_data), lty = 2, col = "red")
plot(1:nDraws, gibbsDraws[,2], type = "l",#col="red",
     xlab = 'Iterations', ylab = 'Posterior Var') # traceplot of Gibbs draws
abline(h = var(ln_data), lty = 2, col = "red")

# par(mfrow=c(2,1))
# Acf for Gibbs draws
barplot(height = a_mu_Gibbs$acf[-1], main = 'Auto correlation for Mean')
barplot(height = a_var_Gibbs$acf[-1], main = 'Auto correlation for Var')

# par(mfrow=c(2,1))
# Histogram of Gibbs draws
hist(gibbsDraws[,1], xlab = 'Posterior Draws of Mean')
hist(gibbsDraws[,2], xlab = 'Posterior Draws of Sigma-Squared')


#Question b: Posterior predictive density vs Actual data
#lnY is Normal
# Applying the posterior parameters to draw posterior predictions
pos_pred_ln <- apply(gibbsDraws, 1,
                     function(x) rnorm(n = 1, mean = x[1], sd = sqrt(x[2])))
plot(density(data), type = "l",col="red",
     xlab = 'Y', ylab = 'Density', main = "Actual vs Posterior Pred Hist",
     ylim = c(0, 0.15))
lines(density(exp(pos_pred_ln)), type = "l",col="blue")
legend("topright", legend=c("Actual Data", "Posterior Prediction"),
       col=c("red", "blue"), lwd=10)

#Get data
data <- read.table('eBayNumberOfBidderData.dat', header = TRUE)
X <- data[,2:ncol(data)]
y <- data[,'nBids']

#Question a: Get the Maximum Likelihood Estimator of beta using glm
glm_model <- glm(formula = nBids ~ ., family = 'poisson',
                 data = subset(data, select = -Const))

print('Maximum likelihood estimators for betas:')
glm_model$coefficients

sign_coeff <- (summary(glm_model)$coefficients[,c(1,4)])
print('Significant covariates:')
sign_coeff[sign_coeff[,2] < 0.05 ,] #p-values less than 0.05 are significant

#Question b: Get beta approximation
#Data - Y/nBids is Poisson[exp(X%*%beta)]
#Prior - beta is Normal[0, 100.solve(XX)]
#Posterior - beta is multivariate normal[posterior mode, neg Hessian at pos mode]

# Functions that returns the log posterior for the Poisson regression.
# First input argument of this function must be the parameters we optimize on,
# i.e. the regression coefficients beta.

LogPostPoisson <- function(betas,y,X,mu,Sigma){
  linPred <- X%*%betas
  logLik <- sum( linPred*y - exp(linPred) - log(factorial(y)) )
  #if (abs(logLik) == Inf) logLik = -20000; # Likelihood is not finite, stear the optimizer away from h
  logPrior <- dmvnorm(t(betas), mu, Sigma, log=TRUE)#For multivariate

  return(logLik + logPrior)
}

# Setting up the prior
X <- as.matrix(X)
Npar <- dim(X)[2]
mu <- rep(0,Npar) # Prior mean vector
Sigma <- 100 * solve(t(X) %*% X) # Prior covariance matrix

# Select the initial values for beta
initVal <- matrix(0,Npar,1)

# The argument control is a list of options to the optimizer optim,
# where fnscale=-1 means that we minimize
# the negative log posterior. Hence, we maximize the log posterior.
OptimRes <- optim(initVal,LogPostPoisson,gr=NULL,y,X,mu,
```

```r
                  Sigma,method=c("BFGS"),control=list(fnscale=-1),hessian=TRUE)

beta_hat <- OptimRes$par
print('The posterior mode is:')
print(beta_hat)

print('Values of  negative inverse of observed information at mode')
solve(-OptimRes$hessian)

# Question c: Simulate from posterior beta using Metropolis Hasting

# Simulate using N as proposal - L8,S8(Metropolis Hastings RW Algo)
RWMSampler <- function(logPostFunc, Nsamples, Npar, ...){
  # The ... is to use any arbitrary logPostFunc as a function object, provided
  # that the first argument of the function is betas.
  #Initialize post beta matrix
  pos_betas <- matrix(data = NA, nrow = 0, ncol = Npar)
  #Step1: Initialize prev beta values
  prev_betas <- matrix(0,Npar,1)
  pos_betas <- rbind(pos_betas, t(prev_betas))

  for (i in 1:Nsamples) {
    # RWM Algorithm
    #Step2: Sample from proposal
    c <- 0.5#Step value
    cov_mat <- solve(-OptimRes$hessian)
    new_betas <- rmvnorm(n = 1, mean = prev_betas, sigma = c * cov_mat)
    #Step3: Get Acceptance Probability, alpha
    pos_prop_den <- logPostFunc(t(new_betas), ...)
    pos_prev_den <- logPostFunc(prev_betas, ...)
    #When you compute the acceptance probability, program the log posterior density
    #Here we take the exp as we have the log of the posterior density
    alpha_temp <- exp(pos_prop_den - pos_prev_den) #posterior density ratio
    alpha <- min(1, alpha_temp)
    #Step4: Accept or reject the new betas
    if (alpha > runif(1)) {
      #Accept the new beta values
      prev_betas <- t(new_betas)
      pos_betas <- rbind(pos_betas, new_betas)
    }else{
      #Beta values does not change
      prev_betas <- prev_betas
    }
  }
  return(pos_betas)
}
Nsamples = 5000
pos_beta_mat <- RWMSampler(logPostFunc = LogPostPoisson,
                           Nsamples = Nsamples,
                           Npar = Npar,
                           y,X,mu,Sigma)
print('Acceptance rate:')
nrow(pos_beta_mat)/Nsamples#Tune c values based on acceptance rate: 25-30%
```

```r
plot(pos_beta_mat[,1], type = "l",col= "red",
     ylab = 'Intercept', main = "Intercept convergence")
acf(pos_beta_mat[,1], lag.max = 500)
plot(pos_beta_mat[,2], type = "l",col="red",
     ylab = 'PowerSeller', main = "PowerSeller convergence")
acf(pos_beta_mat[,2], lag.max = 500)
plot(pos_beta_mat[,3], type = "l",col="red",
     ylab = 'VerifyID', main = "VerifyID convergence")
acf(pos_beta_mat[,3], lag.max = 500)
plot(pos_beta_mat[,4], type = "l",col="red",
     ylab = 'Sealed', main = "Sealed convergence")
acf(pos_beta_mat[,4], lag.max = 500)
plot(pos_beta_mat[,5], type = "l",col="red",
     ylab = 'Minblem', main = "Minblem convergence")
acf(pos_beta_mat[,5], lag.max = 500)
plot(pos_beta_mat[,6], type = "l",col="red",
     ylab = 'MajBlem', main = "MajBlem convergence")
acf(pos_beta_mat[,6], lag.max = 500)
plot(pos_beta_mat[,7], type = "l",col="red",
     ylab = 'LargNeg', main = "LargNeg convergence")
acf(pos_beta_mat[,7], lag.max = 500)
plot(pos_beta_mat[,8], type = "l",col="red",
     ylab = 'LogBook', main = "LogBook convergence")
acf(pos_beta_mat[,8], lag.max = 500)
plot(pos_beta_mat[,9], type = "l",col="red",
     ylab = 'MinBidShare', main = "MinBidShare convergence")
acf(pos_beta_mat[,9], lag.max = 500)
apply(pos_beta_mat, 2, mean)

# Question d: Simulate predictive distribution using MCMC draws of betas
#Given: new X, intercept added as 1
NewX <- matrix(c(1, 1, 0, 1, 0, 1, 0, 1.2, 0.8),
               nrow = 1, ncol = 9)

temp <- exp(pos_beta_mat %*% t(NewX))

# Posterior Predictions
y_pred <- apply(temp, 1, rpois, n = 1)
print('Probability of no bidders in new auction:')
total_bids <-  nrow(pos_beta_mat)
sum(y_pred == 0)/total_bids


####Question 3: Time series model in Stan####
# Question a: Simulate from AR(1)
ar1_sim <- function(phi){
  T <- 300
  mu <- 13
  err_sigma_sq <- 3

  x <- vector(length = T)
  x[1] <- mu
  for (i in 2:T) {
    # The AR(1) Model:
    x[i] <- mu + (phi * (x[i-1] - mu)) +
      rnorm(n = 1, mean = 0, sd = err_sigma_sq)
  }
  return(matrix(data = x, nrow = 1))
}

phi_values <- seq(-1, 1, 0.1)
ar1_sim_lst <- lapply(phi_values, function(phi){
  ar1_sim(phi)
})
#par(mfrow=c(2,1))
plot(seq(1,300),ar1_sim_lst[[1]], type = "l", main = "Phi = -1")
plot(seq(1,300),ar1_sim_lst[[11]], type = "l", col = "red", main = "Phi = 0")
plot(seq(1,300),ar1_sim_lst[[21]], type = "l", col = "red", main = "Phi = 1")

# Question b: Use STAN to sample from posterior
library(rstan)
# Data
phi_values <- c(0.2,0.95)
ar1_sim_lst <- lapply(phi_values, function(phi){
  ar1_sim(phi)
})
library(abind)#To combine a list of matrix into a single matrix
ar1_sim_lmat <- abind(ar1_sim_lst, along = 1)
N <- dim(ar1_sim_lmat)[2]


# Model
StanModel <- '
data {
  int<lower=0> N;
  vector[N] y;
}
parameters {
  real mu;
  real phi;
  real<lower=0> sigma;
}
model {
  //Flat priors are non-informative and need not be defined
  for (n in 2:N)
    y[n] ~ normal(mu + phi * (y[n-1]-mu), sigma);//AR(1) process simulated above
}
'

print("The following is for phi=0.2")
data <- list(N=N, y=ar1_sim_lmat[1,])#For phi=0.2
#data <- list(N=N, y=ar1_sim_lmat[2,])#For phi=0.95
warmup <- 1000
niter <- 2000
fit <- stan(model_code=StanModel,data=data, warmup=warmup,iter=niter,chains=4, verbose = FALSE)
# Print the fitted model
```

```r
#print(fit,digits_summary=3)
print(paste('Posterior mean of mu:',
            summary(fit)$summary[1, "mean"]))
print(paste('Posterior mean of phi:',
            summary(fit)$summary[2, "mean"]))
print(paste('Posterior mean of sigma-sq:',
            summary(fit)$summary[3, "mean"]))

print(paste('95% CI of mu:',
            summary(fit)$summary[1, c("2.5%", "97.5%")]))
print(paste('95% CI of phi:',
            summary(fit)$summary[2, c("2.5%", "97.5%")]))
print(paste('95% CI of sigma-sq:',
            summary(fit)$summary[3, c("2.5%", "97.5%")]))

print(paste('Rhat value for mu:',
            summary(fit)$summary[1, "Rhat"]))
print(paste('Rhat value for phi:',
            summary(fit)$summary[2, "Rhat"]))
print(paste('Rhat value for sigma-sq:',
            summary(fit)$summary[3, "Rhat"]))

# Extract posterior samples
postDraws <- extract(fit)
# Do traceplots of the first chain
par(mfrow = c(3,1))
plot(postDraws$mu[1:(niter-warmup)],type="l",ylab="mu",main="Traceplot")
plot(postDraws$phi[1:(niter-warmup)],type="l",ylab="phi",main="Traceplot")
plot(postDraws$sigma[1:(niter-warmup)],type="l",ylab="sigma",main="Traceplot")
# Do automatic traceplots of all chains
traceplot(fit)
# Bivariate posterior plots
#pairs(fit)

print("The following is for phi=0.95")
# data <- list(N=N, y=ar1_sim_lmat[1,])#For phi=0.2
data <- list(N=N, y=ar1_sim_lmat[2,])#For phi=0.95
warmup <- 1000
niter <- 2000
fit <- stan(model_code=StanModel,data=data, warmup=warmup,iter=niter,chains=4, verbose = FALSE)
# Print the fitted model
#print(fit,digits_summary=3)
print(paste('Posterior mean of mu:',
            summary(fit)$summary[1, "mean"]))
print(paste('Posterior mean of phi:',
            summary(fit)$summary[2, "mean"]))
print(paste('Posterior mean of sigma-sq:',
            summary(fit)$summary[3, "mean"]))

print(paste('95% CI of mu:',
            summary(fit)$summary[1, c("2.5%", "97.5%")]))
print(paste('95% CI of phi:',
            summary(fit)$summary[2, c("2.5%", "97.5%")]))
```

```r
print(paste('95% CI of sigma-sq:',
            summary(fit)$summary[3, c("2.5%", "97.5%")]))

print(paste('Rhat value for mu:',
            summary(fit)$summary[1, "Rhat"]))
print(paste('Rhat value for phi:',
            summary(fit)$summary[2, "Rhat"]))
print(paste('Rhat value for sigma-sq:',
            summary(fit)$summary[3, "Rhat"]))

# Extract posterior samples
postDraws <- extract(fit)
# Do traceplots of the first chain
par(mfrow = c(3,1))
plot(postDraws$mu[1:(niter-warmup)],type="l",ylab="mu",main="Traceplot")
plot(postDraws$phi[1:(niter-warmup)],type="l",ylab="phi",main="Traceplot")
plot(postDraws$sigma[1:(niter-warmup)],type="l",ylab="sigma",main="Traceplot")
# Do automatic traceplots of all chains
traceplot(fit)

pairs(fit)
```