

# Solution to computer exam in Bayesian learning

*Per Siden*

2019-08-21

First load all the data into memory by running the R-file given at the exam

```
rm(list=ls())
source("ExamData.R")
```

```
## Warning: package 'mvtnorm' was built under R version 3.5.2
```

```
set.seed(1)
```

## Problem 1

1a

```
# Prior
nCovs = dim(X)[2]
mu_0 = rep(0, nCovs)
Omega_0 = (1/100)*diag(nCovs)
v_0 = 1
sigma2_0 = 5^2

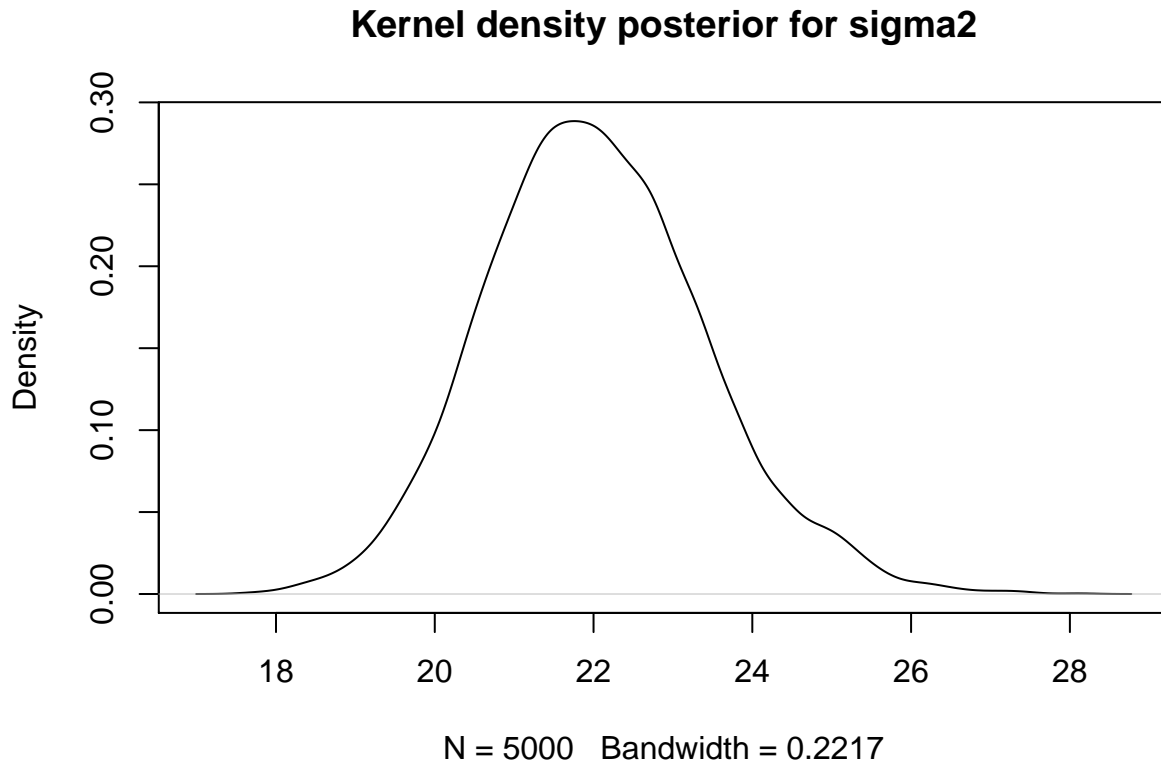
BostonRes <- BayesLinReg(y, X, mu_0, Omega_0, v_0, sigma2_0, nIter = 5000)
Bmean = colMeans(BostonRes$betaSample)
Bq025 = apply(BostonRes$betaSample, 2, quantile, .025)
Bq975 = apply(BostonRes$betaSample, 2, quantile, .975)
print(data.frame(round(cbind(Bmean, Bq025, Bq975), 3), row.names=covNames))
```

##	Bmean	Bq025	Bq975
## intercept	36.043	26.128	45.877
## crim	-0.108	-0.172	-0.047
## zn	0.047	0.020	0.073
## indus	0.019	-0.101	0.140
## chas	2.714	1.070	4.348
## nox	-17.469	-25.041	-10.039
## rm	3.822	3.005	4.638
## age	0.001	-0.025	0.027
## dis	-1.473	-1.853	-1.086
## rad	0.304	0.181	0.428
## tax	-0.012	-0.019	-0.005
## ptratio	-0.943	-1.199	-0.684
## black	0.009	0.004	0.015
## lstat	-0.525	-0.624	-0.419

With 95% probability, a higher nox by 1 part per 10 million is associated with a house price that is between 10 and 25 thousand dollars lower.

1b

```
sigma2Sample = BostonRes$sigma2Sample  
  
dS = density(sigma2Sample)  
par(mfrow=c(1,1))  
plot(dS,main="Kernel density posterior for sigma2")
```



```
dn=sort(dS$y/sum(dS$y),index.return=TRUE);  
dnn = cumsum(dn$x)  
HPDind = sort(dn$ix[dnn > .1])  
HPDCI = c(min(dS$x[HPDind]),max(dS$x[HPDind]))  
cat("Posterior mode: ",dS$x[which.max(dS$y)])
```

```
## Posterior mode: 21.76603
```

```
cat("Highest posterior density interval: ",HPDCI)
```

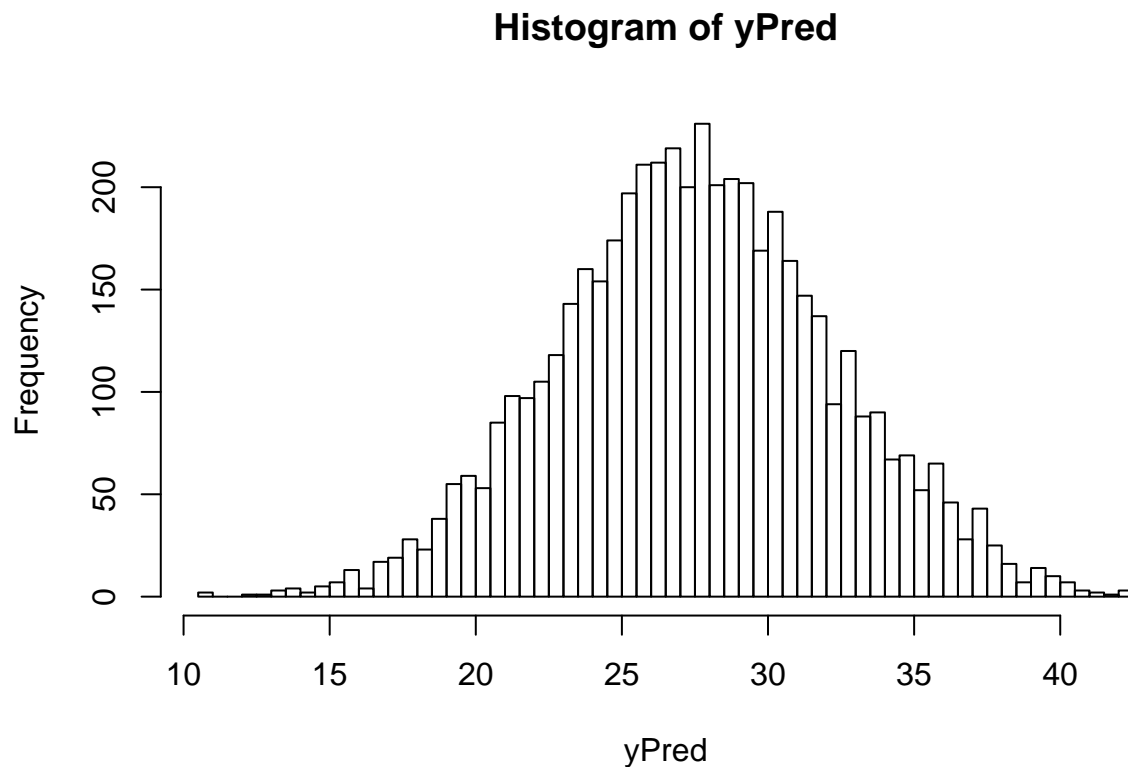
```
## Highest posterior density interval: 19.66839 24.27857
```

1c

What we need here is the predictive distribution for the price given the explanatory variables. We can obtain the predictive distribution by simulation.

```
nSim <- dim(BostonRes$betaSample)[1] # One predictive draw for each posterior parameter draw  
yPred <- matrix(0,nSim,1)  
for (i in 1:nSim){  
  yPred[i] <- XNewHouse%*%BostonRes$betaSample[i,] + rnorm(n = 1, mean = 0, sd = sqrt(BostonRes$sigma2S  
}
```

```
par(mfrow=c(1,1))
hist(yPred,50)
```



```
sum(yPred>=20)/nSim
```

```
## [1] 0.9438
```

Probability of getting \$20000 is quite large (0.94), so the construction project is probably a good idea.

## Problem 2

2a

On paper.

2b

```
x <- c(35,14,4,10,2)
n <- 5
alpha <- 1
beta <- 1

nSim <- 10000
theta <- rbeta(nSim,alpha+n,beta+sum(x))
xPred <- rgeom(nSim,theta)
quantile(xPred,0.95)
```

```
## 95%
```

## 44

The 95% upper bound of the predictive distribution is roughly 44 years.

### Problem 3

3a

On paper.

3b

On paper.

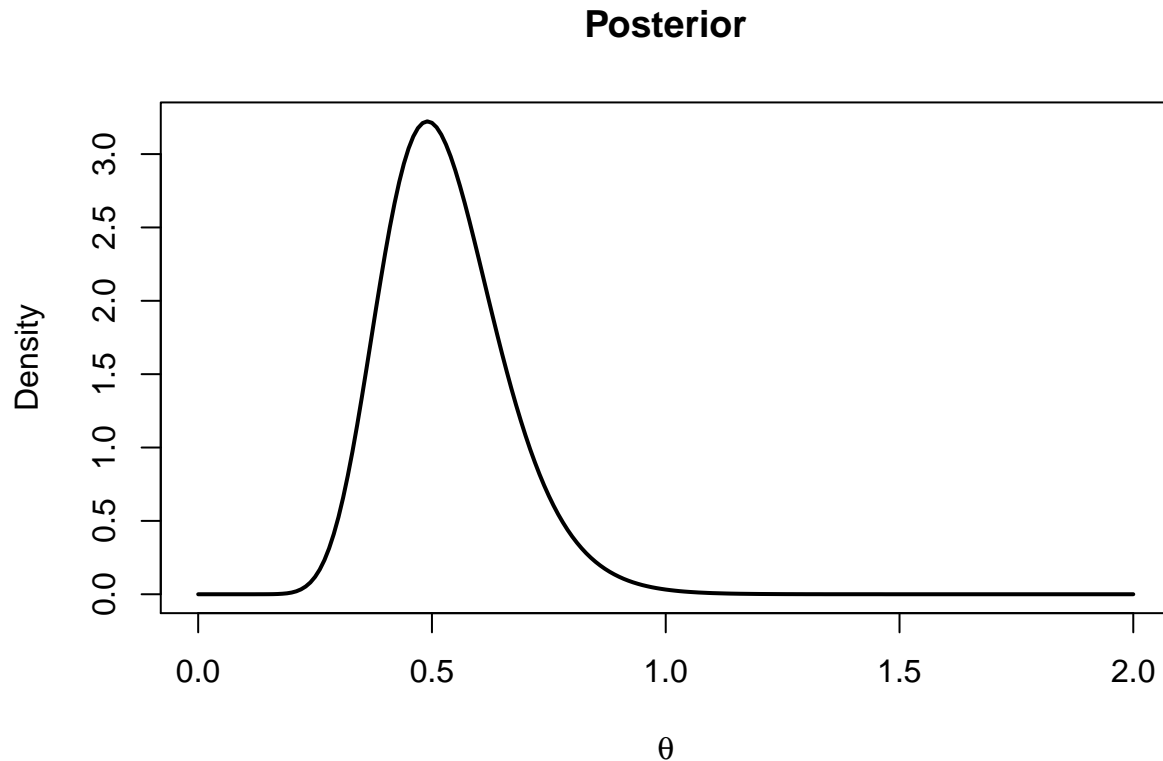
3c

```
# Alternative own implementation of t pdf
logdt <- function(y,theta){
  # lgamma(3) - lgamma(2.5) - .5*log(5*pi) -3*log(1+1/5*(y-log(theta))^2)
}

# Log posterior
logpost <- function(x, y, theta){
  logdens <- sum(dexp(x,rate=theta,log=TRUE)) + sum(dt(y-log(theta),df=5,log=TRUE)) + dgamma(theta,shape=5,rate=1,log=TRUE)
  return(logdens)
}

# Plot the posterior over a grid of theta values
par(mfrow=c(1,1))
gridWidth <- .01
thetaGrid <- seq(0, 2, by = gridWidth)
logPostGrid <- rep(0,length(thetaGrid))
count <- 0
for (theta in thetaGrid){
  count <- count + 1
  logPostGrid[count] <- logpost(xData,yData,theta)
}

# Taking exp() and normalizing so the density integrates to one. Note the (1/gridWidth) factor.
postGrid <- (1/gridWidth)*exp(logPostGrid)/sum(exp(logPostGrid))
plot(thetaGrid, postGrid, type = "l", lwd = 2,main="Posterior",
      ylab = "Density", xlab = expression(theta))
```



## Problem 4

4a

```
x = incidents$incidents
# plot(x)

# Own definition of neg-bin log pdf
logdNegBin <- function(x,mu,phi){
  lchoose(x+phi-1,x) + x*log(mu/(mu+phi)) + phi*log(phi/(mu+phi))
}

logPostNegBin <- function(param, x){
  theta1 = param[1]
  theta2 = param[2]
  logPost = sum(logdNegBin(x, theta1, theta2)) - 2*log(theta2)
  return(logPost)
}

# Code that could be used for normal approximation
# initVal = c(200,20)
# optRes <- optim(par = initVal, fn = logPostNegBin, gr = NULL, x, method = c("L-BFGS-B"),
#               lower = c(0.0001,0.0001), upper = c(Inf,Inf), control = list(fnscale = -1), hessian =
# postMean <- optRes$par # This is the mean vector
# postCov <- -solve(optRes$hessian) # This is posterior covariance matrix

library(mvtnorm)
Metropolis <- function(c,niter,warmup,initVal,Sigma,logPostFunc,x) {
```

```

theta <- initVal
thetamat <- matrix(0,length(theta),warmup+niter)
thetamat[,1] <- theta
accprobvec <- rep(0,warmup+niter)

for(i in 2:(warmup+niter)) {
  thetaProp <- t(rmvnorm(1,theta,c*Sigma))
  thetaProp[thetaProp <= 0] = 1e-6
  accprob <- exp(logPostFunc(thetaProp,x) - logPostFunc(theta,x))
  accprobvec[i] <- min(accprob,1)
  if(runif(1) < accprob) {
    theta <- thetaProp
  }
  thetamat[,i] <- theta
}
return(list(thetamat=thetamat,accprobvec=accprobvec))
}

c <- .1
niter <- 1000
warmup <- 50

initVal = c(200,20)
Sigma = c*diag(c(100,5))

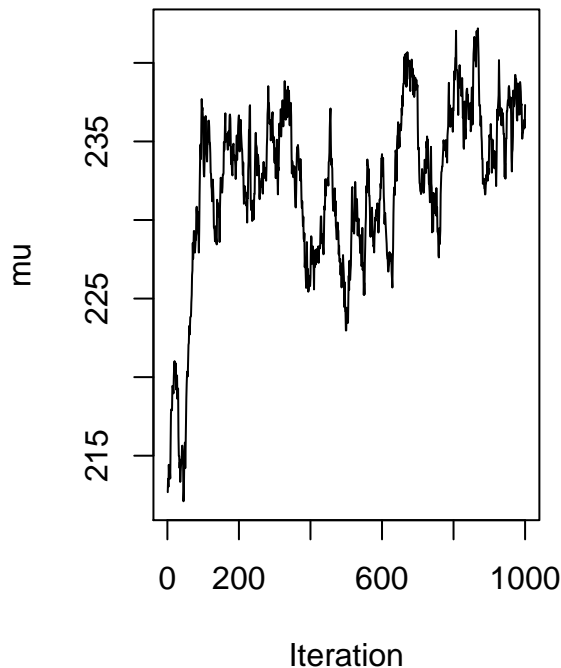
mp <- Metropolis(c,niter,warmup,initVal,Sigma,logPostNegBin,x)

# Optionally compute posterior mean and variance
# rowMeans(mp$thetamat[,warmup+1:niter])
# apply(mp$thetamat,1,var)

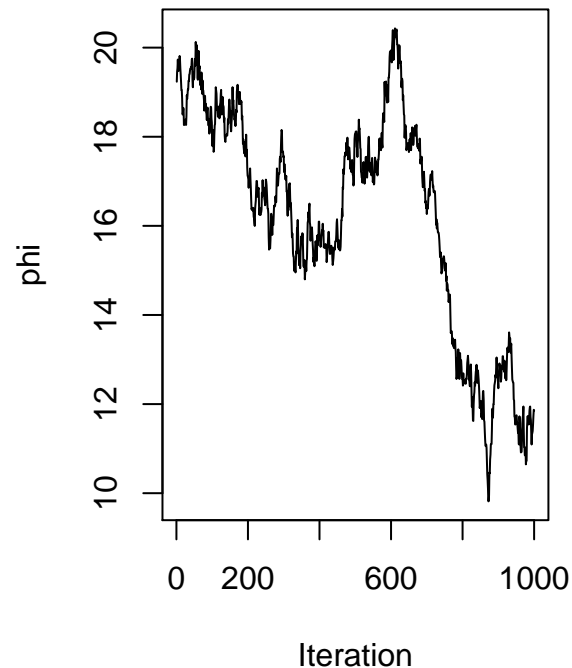
par(mfrow=c(1,2))
plot(mp$thetamat[1,warmup+1:niter], type="l",main="Traceplot",xlab="Iteration",ylab="mu")
plot(mp$thetamat[2,warmup+1:niter], type="l",main="Traceplot",xlab="Iteration",ylab="phi")

```

Traceplot



Traceplot



```
mean(mp$accprobvec)
```

```
## [1] 0.9333279
```

The traceplots show that the parameters move slowly in the posterior space which means that the sampler is not efficient. This can also be seen in that the acceptance rate is high (around 90%) and should optimally be around 25-30% for random walk Metropolis. The efficiency could be improved by increasing the value of  $c$  to make more distant proposals.

#### 4b

Function for proposing from the independent multivariate gamma distribution

```
rmvgamma <- function(theta,c){
  return(c(rgamma(1,shape=c*theta[1],rate=c),rgamma(1,shape=c*theta[2],rate=c)))
}
```

Function to compute the log proposal density ratio for the Metropoli-Hastings acceptance probability

```
ldmvgamma <- function(theta,thetaProp,c){
  ldens = dgamma(theta[1],shape=c*thetaProp[1],rate=c,log=TRUE) +
    dgamma(theta[2],shape=c*thetaProp[2],rate=c,log=TRUE) -
    dgamma(thetaProp[1],shape=c*theta[1],rate=c,log=TRUE) -
    dgamma(thetaProp[2],shape=c*theta[2],rate=c,log=TRUE)
  return(ldens)
}
```

```
c = .8
```

```
MetropolisHastings <- function(c,niter,warmup,initVal,logPostFunc,x) {
  theta <- initVal
```

```

thetamat <- matrix(0,length(theta),warmup+niter)
thetamat[,1] <- theta
accprobvec <- rep(0,warmup+niter)

for(i in 2:(warmup+niter)) {
  thetaProp <- rmvgamma(theta,c)
  accprob <- exp(logPostFunc(thetaProp,x) - logPostFunc(theta,x) + ldmvgamma(theta,thetaProp,c))
  accprobvec[i] <- min(accprob,1)
  if(runif(1) < accprob) {
    theta <- thetaProp
  }
  thetamat[,i] <- theta
}

return(list(thetamat=thetamat,accprobvec=accprobvec))
}

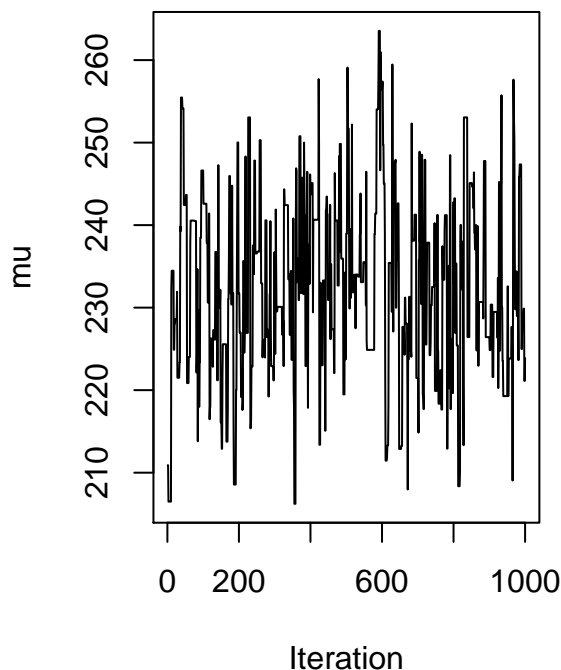
mp <- MetropolisHastings(c,niter,warmup,initVal,logPostNegBin,x)

# Optionally compute posterior mean and variance
# rowMeans(mp$thetamat[,warmup+1:niter])
# apply(mp$thetamat,1,var)

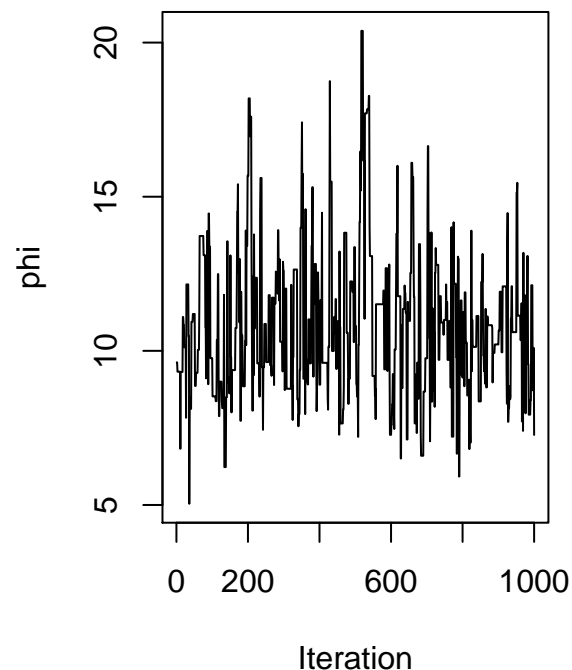
par(mfrow=c(1,2))
plot(mp$thetamat[1,warmup+1:niter], type="l",main="Traceplot",xlab="Iteration",ylab="mu")
plot(mp$thetamat[2,warmup+1:niter], type="l",main="Traceplot",xlab="Iteration",ylab="phi")

```

**Traceplot**



**Traceplot**



```
mean(mp$accprobvec)
```

```
## [1] 0.3591745
```



The sampler seems to rapidly explore the posterior distribution so it is quite efficient.