Bayesian Learning Assignment A

Thomas Mortensen, Julia Langenius, Alex Holmskär

2025-09-17

Problem 1

1a)

Due to the conjugate prior our posterior will also be beta distributed and we know the values for its parameters and expressions for mean and variance.

$$\alpha_{posterior} = \alpha_0 + s = 2 + 14 = 16$$

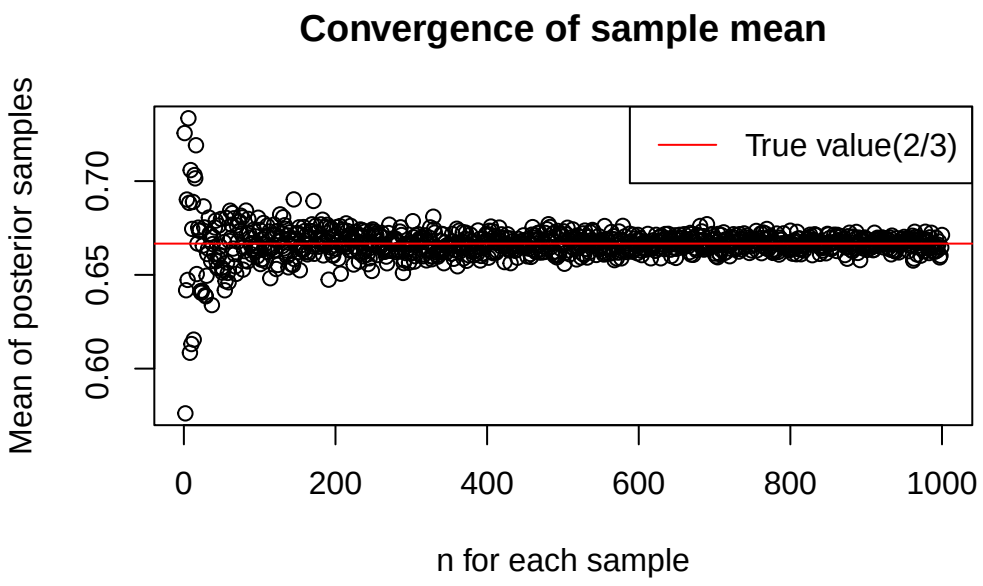
$$\beta_{posterior} = \beta_0 + f = 2 + 6 = 8$$

$$E(\theta|y) = \frac{\alpha_{posterior}}{\alpha_{posterior} + \beta_{posterior}} = \frac{16}{16+8} = \frac{16}{24} = \frac{2}{3}$$

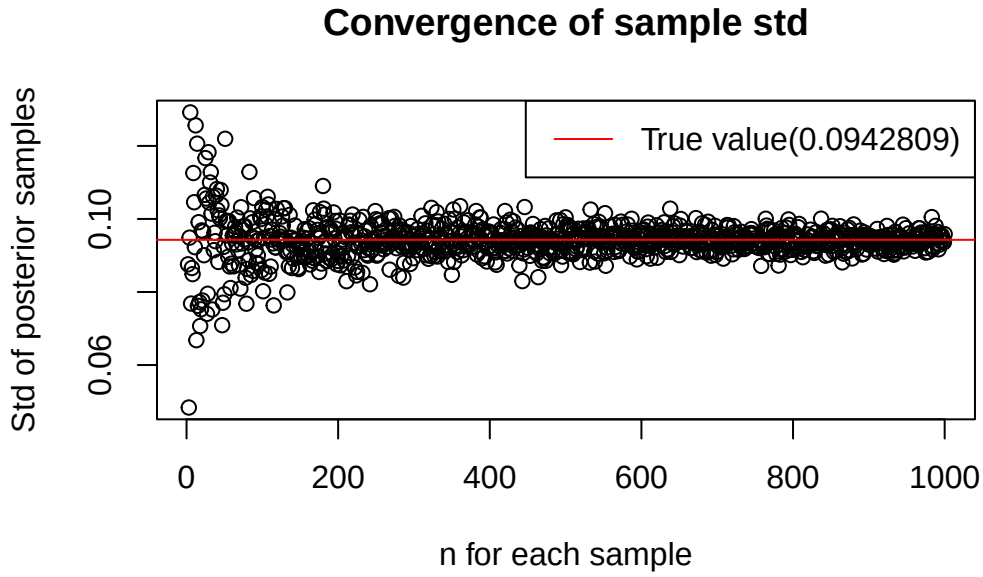
$$\sqrt{V(\theta|y)} = \sqrt{\frac{\alpha_{posterior}\beta_{posterior}}{(\alpha_{posterior} + \beta_{posterior} + 1)(\alpha_{posterior} + \beta_{posterior})^2}} = \sqrt{\frac{16 \times 8}{(25)(24)^2}} = \sqrt{\frac{128}{14400}} \approx 0.0942809$$

Here we can see the LLN at work as the sample quantities based on the sample mean, converge to their population counterparts.

```
set.seed(123)
##| echo: false
plot(sapply(1:1e3, function(l) mean(rbeta(l, 16, 8))), ylab = "Mean of posterior samples", xl

abline(h = 2/3, col = "red")
legend("topright", legend = "True value(2/3)", col = "red", lty = 1, lwd = 1)
```




```r
plot(sapply(1:1e3, function(l) sd(rbeta(l, 16, 8))), ylab = "Std of posterior samples", xlab
abline(h = 0.0942809, col = "red")
legend("topright", legend = "True value(0.0942809)", col = "red", lty = 1, lwd = 1)
```



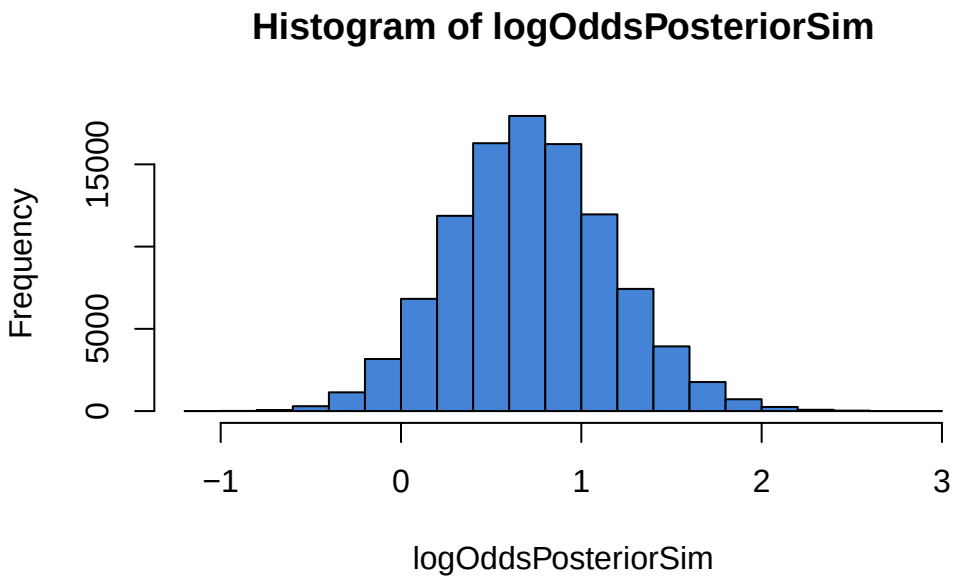
1b)

```r
set.seed(101)
ndraws = 1e4
PosteriorMCsample = rbeta(ndraws,16,8)
probEstimate = mean(PosteriorMCsample < 0.5)
data.frame(pTrue = pbeta(0.5, 16,8), pEstimate = probEstimate, row.names = "Probabilities")
```

```
                   pTrue pEstimate
Probabilities 0.04656982    0.0466
```

1c)

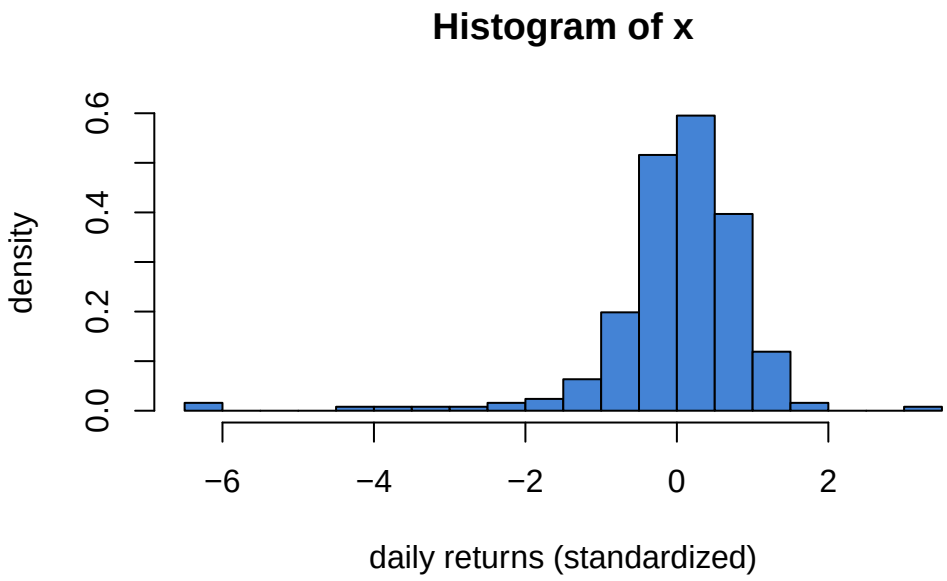
```r
logodds = function(prob){
  return( log(prob / (1-prob)) )
}
logOddsPosteriorSim = logodds(rbeta(n = 1e5, 16, 8))

hist(logOddsPosteriorSim, col = "#4483d5")
```



Problem 2

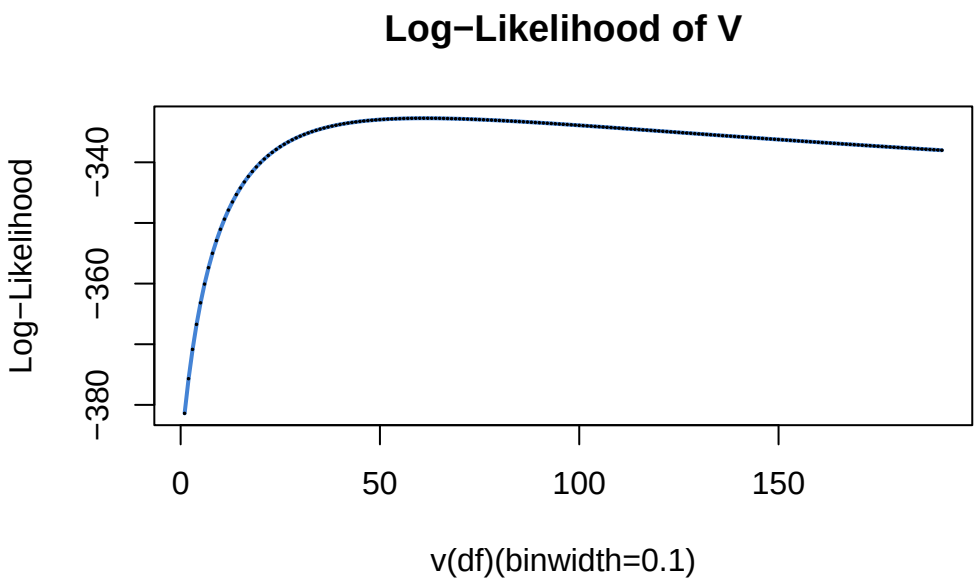
2a)

```
load("ericsson.Rdata")

x = (returns - mean(returns)) / sd(returns) # standardized returns


hist(x, 30, freq = FALSE, xlab = "daily returns (standardized)", ylab = "density", col = "#4
```



V = 7 seems visually to be the MLE of V.

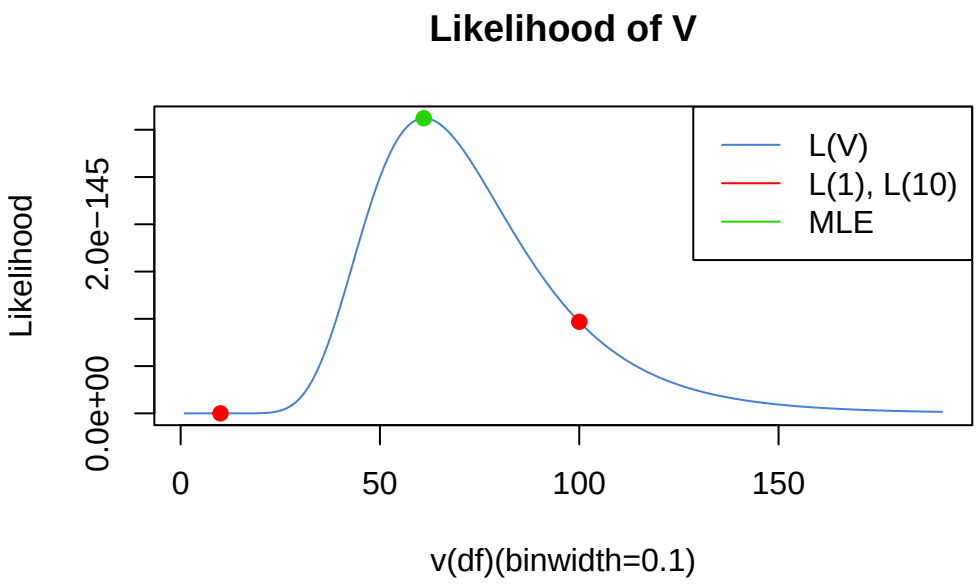
```
binwidth = 0.1
vGrid = seq(1,20, by = binwidth)
vLogLik = sapply(vGrid, function(v) sum(log(dt(x, v))))
plot(vLogLik, ylab = "Log-Likelihood", xlab = paste0("v(df)(binwidth=", binwidth,")"), main =
points(vLogLik, cex = 0.1)
```



2b)

We can see that both graphically and numerically, v = 10 has a higher likelihood than v = 1. This makes sense as our MLE of 7 is closer to 10 and more of the density is around 10 than 1.

```
vLik = sapply(vGrid, function(v) prod((dt(x, v))))
plot(vLik, ylab = "Likelihood", xlab = paste0("v(df)(binwidth=", binwidth,")"), main = "Likel
points(61, vLik[61], col = "#23d600", pch = 19)
legend("topright", col = c("#4483d5", "red", "#23d600"), legend = c("L(V)", "L(1), L(10)", "M

#plot(sapply(1:100, function(v) exp(sum(log(dt(x, v))))), ylab = "Log-Likelihood", xlab = "v
#lines(sapply(1:100, function(v) exp(sum(log(dt(x, v))))))
points(c(1/binwidth,10/binwidth),vLik[c(1/binwidth,10/binwidth)], col = "red", pch = 19)
```

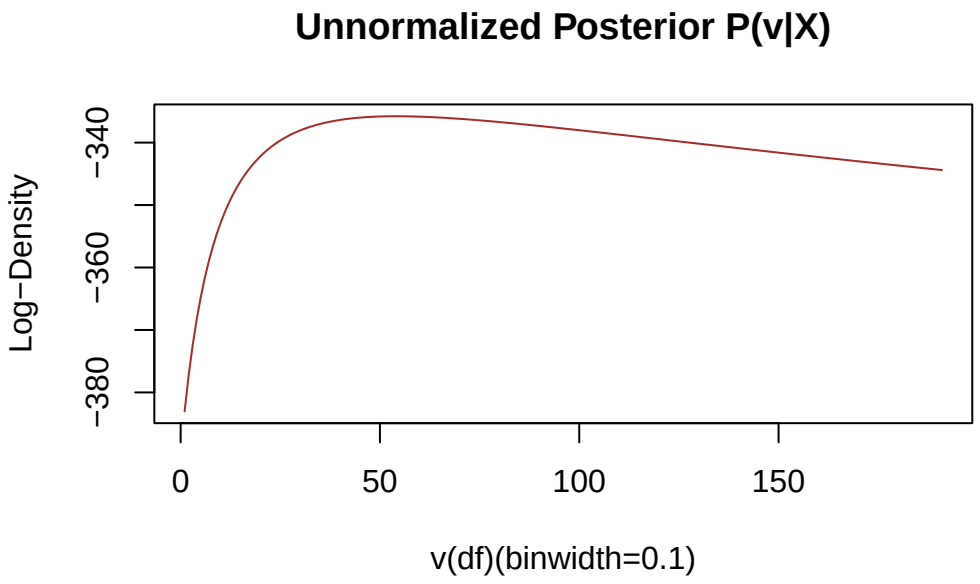



```
cat("L(1) and L(10) are:",vLik[c(1,10)],"MLE: 7")
```

```
L(1) and L(10) are: 2.296528e-166 3.515496e-153 MLE: 7
```

2c)

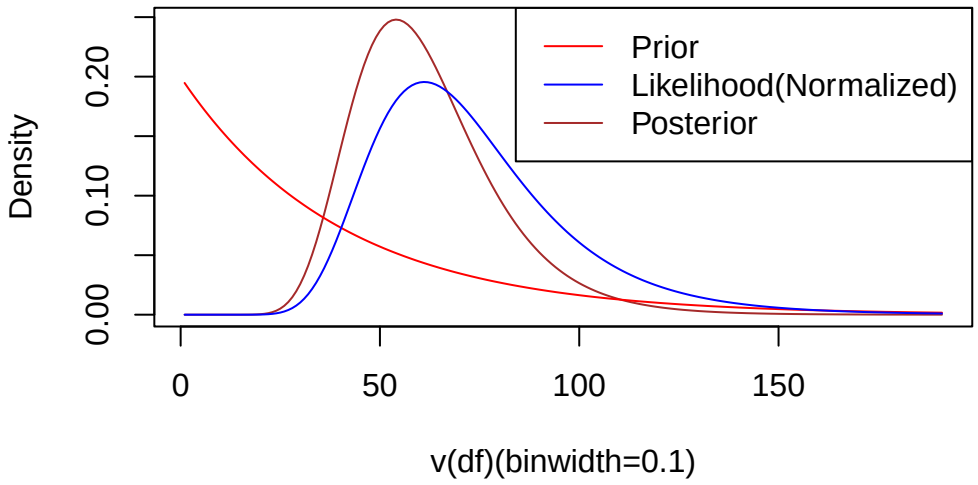
```
vLogPrior = dexp(vGrid, rate = 0.25, log = TRUE)

vLogPosterior = vLogLik + vLogPrior

plot(vLogPosterior, xlab = paste0("v(df)(binwidth=", binwidth,")"), ylab = "Log-Density", ma
```




```
#lines(vLogPosterior)
```

2d)

```
vPosterior = exp(vLogPosterior)
vPosteriorNorm = vPosterior / (sum(vPosterior)*binwidth)

# logsumexp <- function(log_values) {
#     m <- max(log_values)                # find max
#     m + log(sum(exp(log_values - m)))   # its called the logsumexp trick for these types o
# })
# logZ = logsumexp(vLogPosterior + log(dv))
# vPosteriorNorm = exp(vLogPosterior - logZ)


plot(vPosteriorNorm, xlab = paste0("v(df)(binwidth=", binwidth,")"), ylab = "Density", type =
#lines(vPosteriorNorm)
lines(exp(vLogPrior), col = "red")
lines(vLik / sum(vLik*binwidth), col = "blue")
legend("topright", legend = c("Prior","Likelihood(Normalized)","Posterior"), lty = 1, lwd = 
```



2e)

Here is our estimate of the posterior mean through numerical integration, basically using a sum to approximate the integral of $\int P(V \mid X) \times V$.

```
sum(vPosteriorNorm * vGrid * binwidth) # Basically approximate the integral x*f(x) with a sum
```

```
[1] 7.079766
```

Problem 3

You are the manager of a small fruit shop that sells a particular exclusive mango fruit. You buy each mango for $10 and sell them for $20. One reason for the large mark-up is that some of the mango may go unsold, and must then be used for mango smoothies which only brings in $3 per mango, i.e. a loss of $7 on each mango that goes unsold. Each week you must decide on how many mangoes to bring into the shop, and the demand is uncertain.

3a)

As this is a conjugate prior it can be shown that the posterior is also gamma distributed with the following parameters

$$p(\lambda|x) \sim gamma(\alpha = \alpha_{prior} + \sum x_i, \beta = n + \beta_{prior})$$

With a mean and variance formulas such as:

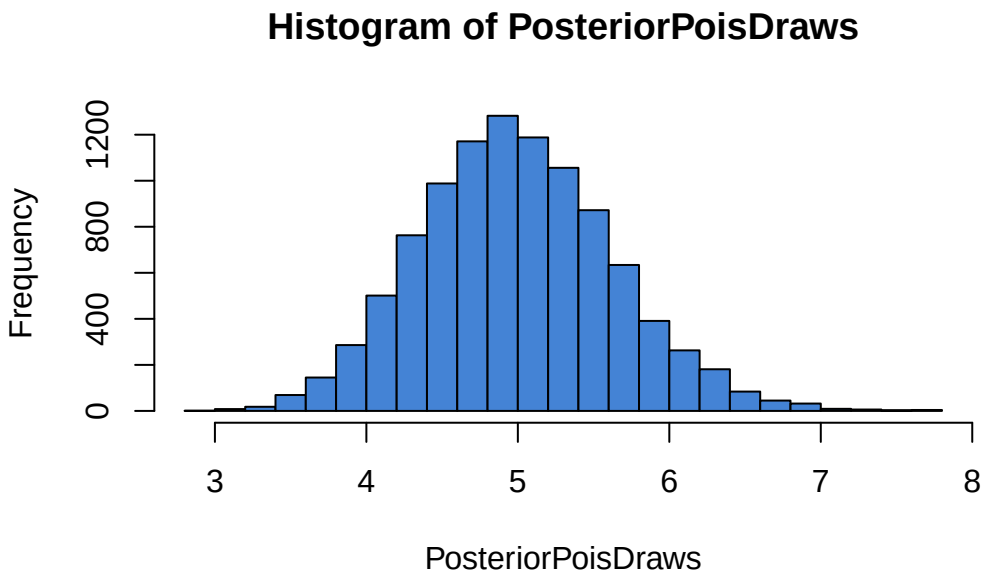
$$E(\lambda|x) = \frac{\alpha}{\beta}$$

$$V(\lambda|x) = \frac{\alpha}{\beta^2}$$

$$\alpha_{prior} = 7, \beta_{prior} = 2$$

Here below we get an estimated probability of $\theta > 8$ by the proportion of values above 8 drawn from the posterior. Sadly as the true probability is about one in 30k, we don't get even one value above 8 from 10k draws so our proportion is 0.

```
set.seed(123)
xMangoes = c(3, 5, 4, 3, 6, 8, 6, 1, 14, 3)
n = length(xMangoes)
alpha_posterior = 7 + sum(xMangoes)
beta_posterior = n + 2
PosteriorPoisDraws = rgamma(n = 1e4, shape = alpha_posterior,rate = beta_posterior) # Shape =

hist(PosteriorPoisDraws, col = "#4483d5", breaks = 20)
```

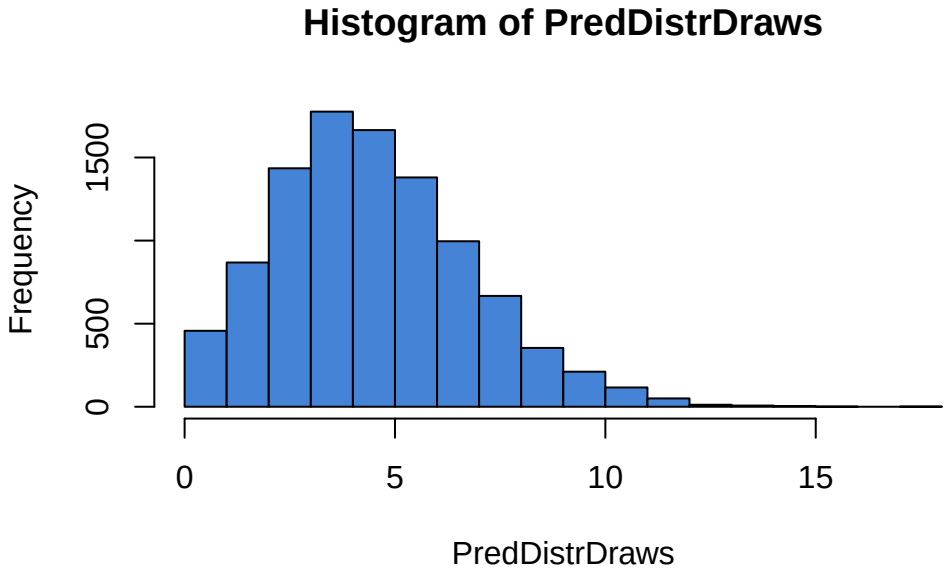



```
ProbEst = mean(PosteriorPoisDraws > 8)
ProbTrue = pgamma(8, shape = alpha_posterior, rate = beta_posterior, lower.tail = FALSE)

data.frame(Probtrue = ProbTrue, ProbEst = ProbEst, row.names = "Probabilities(gamma)")
```

```
                           Probtrue ProbEst
Probabilities(gamma) 3.291869e-05       0
```

3b)

```
PredDistrDraws = rnbinom(n = 1e4, size = alpha_posterior, prob = beta_posterior / (beta_poste

hist(PredDistrDraws, col = "#4483d5", breaks = 20)
```



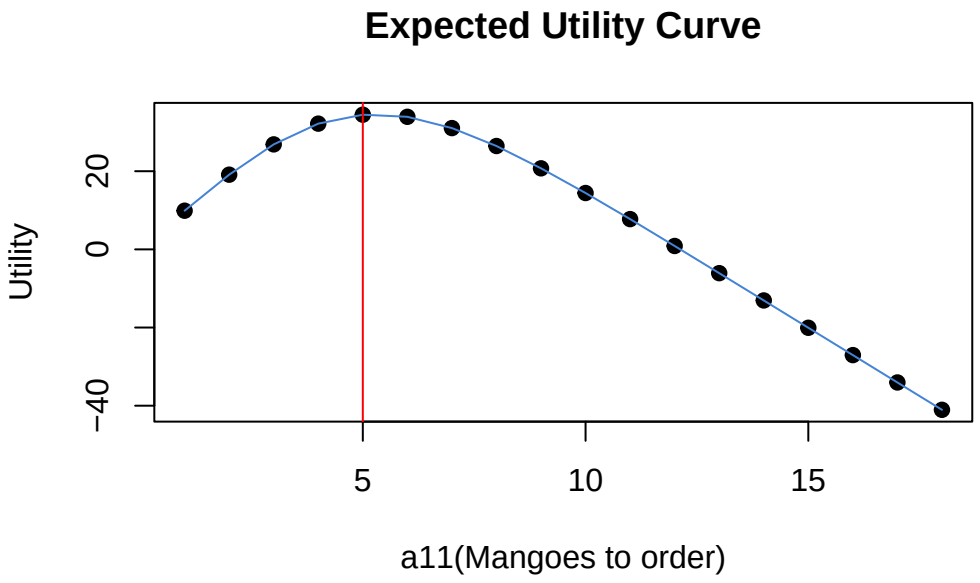

```
ProbDistrEst = mean(PredDistrDraws >= 8)
ProbDistrTrue = pnbinom(7, size = alpha_posterior, prob = beta_posterior / (beta_posterior +
data.frame(ProbTrue = ProbDistrTrue, ProbEst = ProbDistrEst, row.names = "Probabilities(Predi
```

```
                                    ProbTrue ProbEst
Probabilities(Predictive distribution): 0.141594  0.1423
```

3c)

Maximizing Mango utility

```r
utilityFunc = function(x11, a11){
  uVec = vector(length = length(x11))
  j = 1
  for(i in x11){
    uVec[j] = 10*min(i, a11) -7*max(0,a11-i)
    # print(uVec[j])
    # print(j)
    j = j + 1
  }
  return((uVec))
}

a11Grid = 1:max(PredDistrDraws)


UtilDraws = sapply(a11Grid, function(a11) mean(utilityFunc(PredDistrDraws, a11))) # A way of
plot(UtilDraws, xlab = "a11(Mangoes to order)", ylab = "Utility", pch = 19, main = "Expected
lines(UtilDraws, col = "#4483d5")
abline(v = 5, col = "red")
```




```r
cat("The argmax of our expected utility function is =", which.max(UtilDraws))
```

```
The argmax of our expected utility function is = 5
```

Problem 4

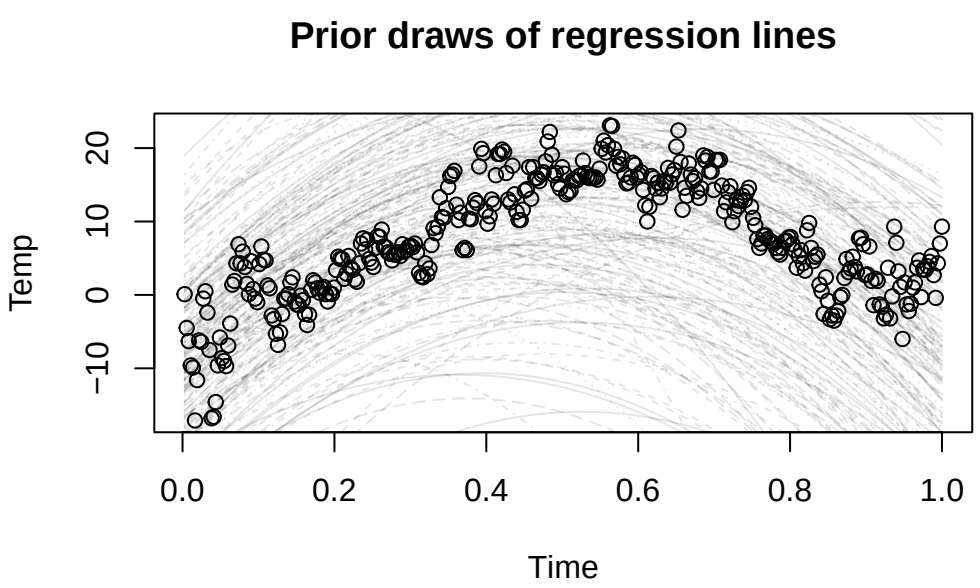
4a)

```
#install.packages("remotes")                # Uncomment this the first time
library(remotes)
#install_github("StatisticsSU/SUdatasets")  # Uncomment this the first time
library(SUdatasets)
head(tempLinkoping)
```

```
         time  temp
1 0.002732240   0.1
2 0.005464481  -4.5
3 0.008196721  -6.3
4 0.010928962  -9.6
5 0.013661202  -9.9
6 0.016393443 -17.1
```

```
library(mvtnorm)
timeVar = as.matrix(tempLinkoping["time"])
tempVar = as.matrix(tempLinkoping["temp"])
xtime = t(cbind(rep(1, 6), timeVar, timeVar^2))
# Simulator for the scaled inverse Chi-square distribution
rScaledInvChi2 <- function(n, v_0, sigma2_0){
  return((v_0*sigma2_0)/rchisq(n, df = v_0))
}
n = 1
v_0 = 3
sigma2_0 = 1
nDrawsMVT = 200
betaDraws = function(iter){
sigma = rScaledInvChi2(n, v_0, sigma2_0)
mu0 = c(-10,100,-100)
omega0_inv = solve(0.01*diag(c(1,1,1)))
beta = rmvnorm(n, mean = mu0 ,sigma = sigma * omega0_inv)
return(beta)
}

set.seed(123)
```

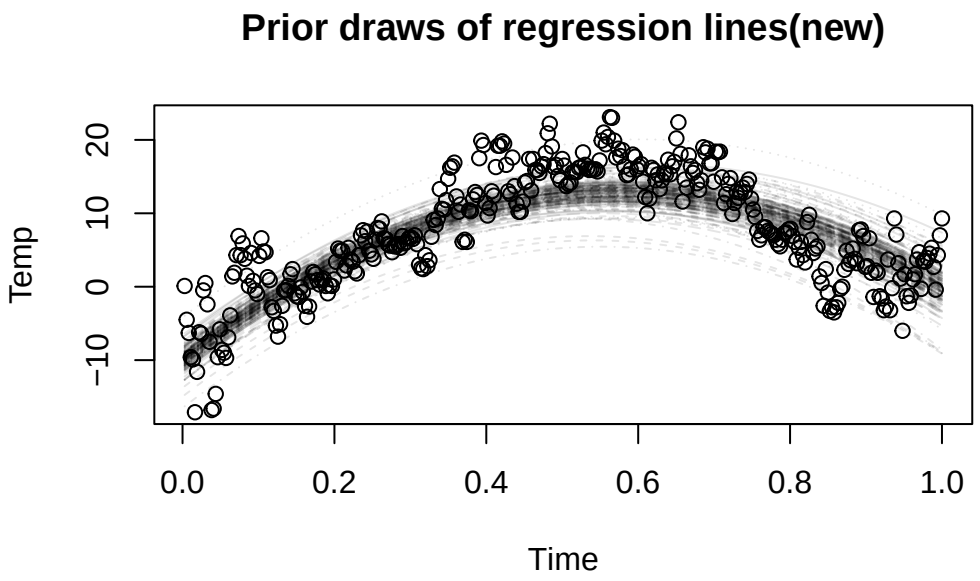
```r
betaVec = t(sapply(1:nDrawsMVT, betaDraws))

predVec = betaVec %*% xtime  # A vector where each row is each datapoint(intercept, x, x^2) t

tGrid = seq(min(timeVar), max(timeVar), length.out = nrow(tempLinkoping))

plot(tGrid, tempVar[,1], xlab = "Time", ylab = "Temp", main = "Prior draws of regression line
matlines(tGrid, t(predVec), col = rgb(0,0,0,0.1))
```



These values for the prior and hyperprior's(hierarchical prior) hyperparameters seems a lot better with more regression lines following/fitting the data better and with less spread in distance and shape.

```r
n = 1
v_0 = 3
sigma2_0 = 3
nDrawsMVT = 200
mu0 = c(-10,80,-70)
omega0 = diag(c(4,4,4))
omega0_inv = solve(omega0)
betaDraws = function(iter){
sigma = rScaledInvChi2(n, v_0, sigma2_0)
beta = rmvnorm(n, mean = mu0 ,sigma = sigma * omega0_inv)
return(beta)
}
```

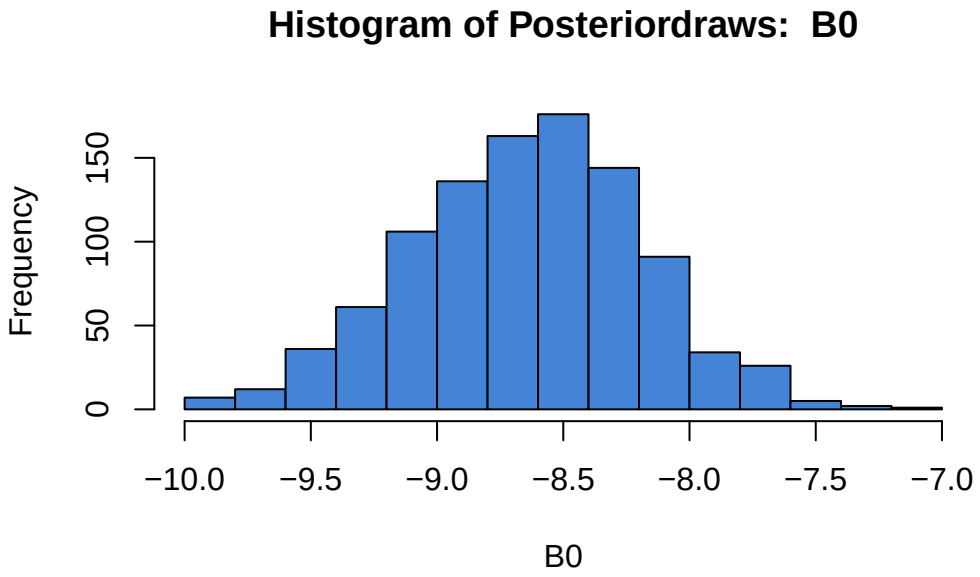
```r
set.seed(123)

betaVec = t(sapply(1:nDrawsMVT, betaDraws))

predVec = betaVec %*% xtime  # A vector where each row is each datapoint(intercept, x, x^2) t

tGrid = seq(min(timeVar), max(timeVar), length.out = nrow(tempLinkoping))

plot(tGrid, tempVar[,1], xlab = "Time", ylab = "Temp", main = "Prior draws of regression line
matlines(tGrid, t(predVec), col = rgb(0,0,0,0.1))
```

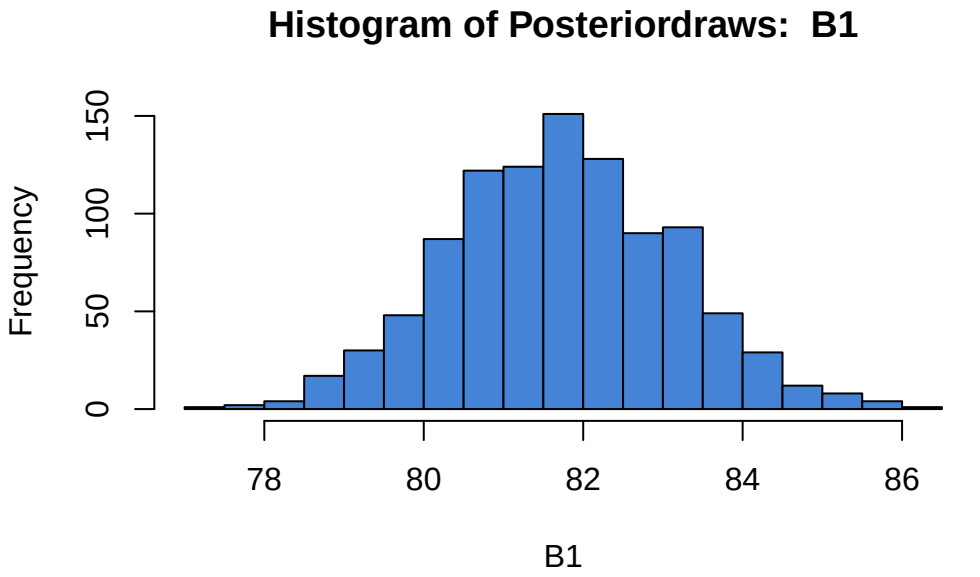


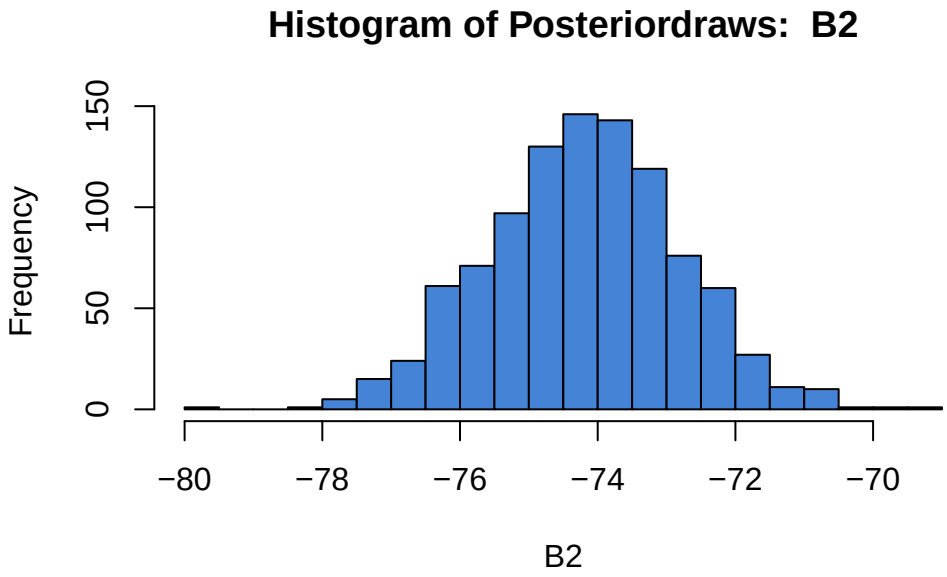
4b)

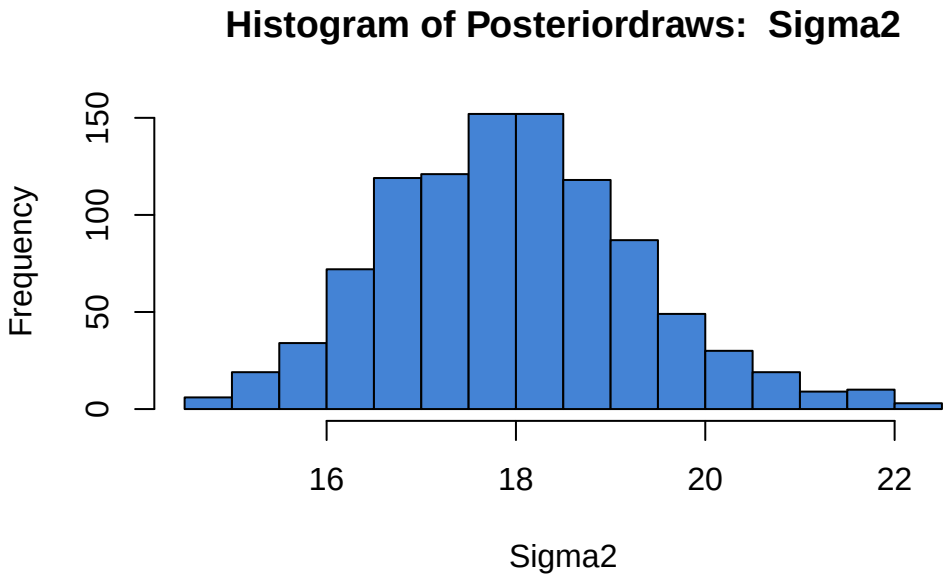
```r
# Page 86 Villani Book for reference. Basically we just sample from the posterior as its form

MvnInvChisqPosteriorSampler = function(nDraws){
n = nrow(tempVar)
p = 3 # b0 + b1 + b2
xtx = xtime %*% t(xtime)
omega_n = xtx + omega0
v_n = v_0 + n
y = as.matrix(tempVar)
betaHat = solve(xtime %*% t(xtime)) %*% xtime %*% y
```

```r
mu_n = solve(omega_n) %*% (xtx %*% betaHat + omega0 %*% mu0)

myDraws = list()

omega_n_inv = solve(omega_n)

myDrawsMat = matrix(NA, nrow = nDraws, ncol = 4)

for(i in 1:nDraws){
  s2 = t(y - t(xtime) %*% betaHat) %*% (y - t(xtime) %*% betaHat) / (n-p)
  v_nSigma2n =  v_0*sigma2_0 + (n-p)*s2 + t(mu_n - betaHat) %*% xtx %*% (mu_n -   betaHat) +
  Sigma2n = v_nSigma2n / v_n

  sigma2 = rScaledInvChi2(n = 1, v_0 = v_n, sigma2_0 = Sigma2n)
  sigma2 = as.numeric(sigma2)
  beta = rmvnorm(1, mean = mu_n ,sigma = sigma2 * omega_n_inv)
  betaHat = t(beta)
  myDrawsMat[i,] =  c(beta, sigma2)
}
myDraws = as.data.frame(myDrawsMat)
colnames(myDraws) = c("B0", "B1", "B2", "Sigma2")
return(myDraws)
}

set.seed(123)

PosteriorDraws = MvnInvChisqPosteriorSampler(nDraws = 1e3)

for(i in 1:4){
  hist(PosteriorDraws[,i], main = paste("Histogram of Posteriordraws: ", colnames(PosteriorDr
}
```


Histogram of Posteriordraws: B0
Frequency
0
50
100
150
−10.0
−9.5
−9.0
−8.5
−8.0
−7.5
−7.0
B0


Histogram of Posteriordraws: B1
Frequency
0
50
100
150
78
80
82
84
86
B1



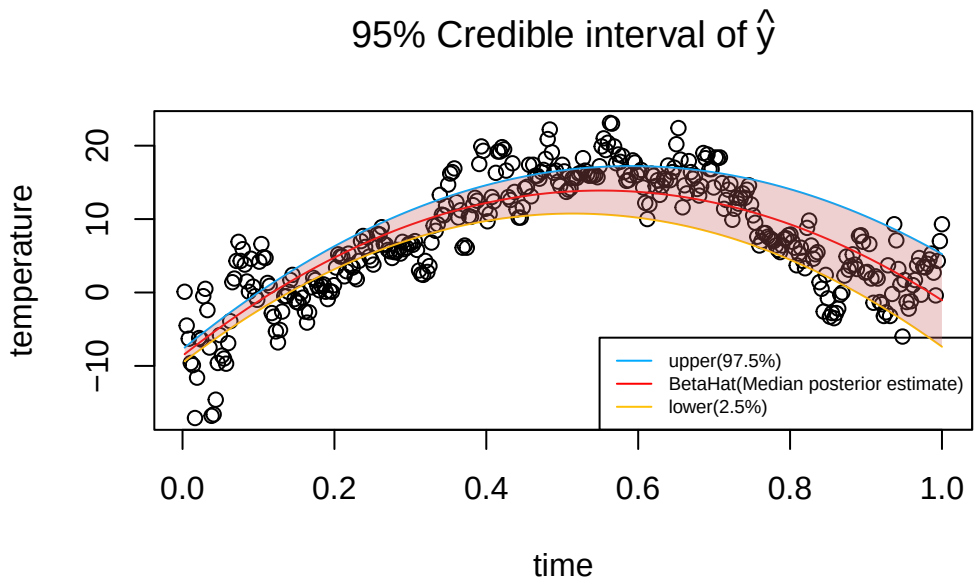



```
medianBetaHat = sapply(1:3, function(j) median(PosteriorDraws[,j]))
medianPreds = medianBetaHat %*% xtime
```

The interval bands do indeed not cover most of the points but this is due to them displaying the interval for the $\hat{\beta}$, having a 95% probability of containing the true $\hat{\beta}$, not a 95% probability of containing the data.

```
CredibleInterval = sapply(1:3, function(i) quantile(PosteriorDraws[,i], probs = c(0.025, 0.9
plot(timeVar, tempVar, xlab = "time", ylab = "temperature", main = expression("95% Credible
```

```r
lines(timeVar,as.numeric(medianPreds), col = "red")
legend("bottomright", legend = c("upper(97.5%)", "BetaHat(Median posterior estimate)", "lowe
lower = CredibleInterval[1,]
upper = CredibleInterval[2,]
lines(timeVar, lower, col = "#ffc107")
lines(timeVar, upper, col = "#00abff")

polygon(
  x = c(timeVar, rev(timeVar)),
  y = c(lower,    rev(upper)),
  col = rgb(0.8, 0.4, 0.4, 0.3),    # R,G,B in [0,1]; alpha=0.3 = 30% opacity
  border = NA                       # no border around the polygon
)
```



4c)

We also checked the highest observed time to be sure and indeed the formula given below from the assignment gives an even higher function value so 0.5507938 does indeed seem to be the time with the highest expected temperature. Here we did it using the $\hat{\beta}$ from the mean of the posterior but the median was also tested and gave the same result.

$$x_{max} = -\frac{\beta_1}{2\beta_2}$$

```r
meanBetaHat = sapply(1:3, function(j) mean(PosteriorDraws[,j]))
meanPreds = meanBetaHat %*% xtime

timeObs = timeVar[which.max(meanPreds)]
ObsExpectedTemp = c(meanBetaHat %*% c(1, 0.5519126, 0.5519126^2))

timeFormula = -(meanBetaHat[2] / (2*meanBetaHat[3]))
FormulaExpectedTemp = c(meanBetaHat %*% c(1, 0.5508196, 0.5508196^2))

data.frame(timeObs, timeFormula, ObsExpectedTemp, FormulaExpectedTemp)
```

```
    timeObs timeFormula ObsExpectedTemp FormulaExpectedTemp
1 0.5519126   0.5507938        13.86675             13.86684
```

```r
cat("The time with the highest expected temperature was: ", timeFormula, " Or day: ", timeFo
```

```
The time with the highest expected temperature was:  0.5507938  Or day:  201.5905
```