Bayesian Learning - Computer Lab 3

Stefano Toffol (steto820) and Nahid Farazmand (nahfa911)

17 May, 2019

Question 1 - Mixture of normal models

We are asked to analyse the dataset `rainfall.dat`, a collection of daily precipitations (rain, hail, snow, ...) from the beginning of 1948 to the end of 1983 at Snoqualmie Falls, Washington. The dataset consists of just one column, containing the observed daily precipitations. The measurements were made in $\frac{1}{100}$ inches.
We will need to fit different models to the dataset and then compare them (graphically).

(a) Normal model

We will assume that each daily precipitation $\{y_1, \ldots, y_n\}$ are independent normally distributed: $y_1, \ldots, y_n | \mu, \sigma^2 \sim \mathcal{N}(\mu, \sigma^2)$.
The priors for the (unknown) parameters are: $\mu \sim \mathcal{N}(\mu, \sigma^2)$ and $\sigma^2 \sim$ Inv-$\chi^2(\nu_0, \sigma_0^2)$, one independent from another.

In order to get an idea of the type of data presented, after a brief research we stumbled across the climate webpage of Snoqualmie Falls from the website `climate-data.org`. The reported average annual precipitations is 2127 mm. Since our data is measure in $\frac{1}{100}$ inches and refers to the daily precipitations, we had to transform the information received. After dividing the value by 365 (the days of the year), transform it in inches and scaling it correctly we got an average daily precipitation of 22.94 approximately, which will be our prior parameter μ_0.
Precipitations is a quite variable phenomena and we therefore expect a high variability. We considered $\tau_0^2 = 25$ a good guess for the variance.
Regarding the parameters of the variance σ^2, we believe our data to be accurate and reliable. However we previously did not consider that we lack of the days with precipitations equal to zero, meaning that our prior value for μ_0 will most likely underestimate the real value. Therefore, being unsure about the actual values of the parameters, we will set $\nu_o = 30$ and $\sigma_0^2 = 1$.

In order to sample from the posterior, we will implement a Gibbs sampler. In Gibbs sampling algorithm, in each iteration we draw from full conditional distribution of each unknown parameter given all other unknown parameters. Formulas of full conditional parameters for μ and σ^2 are as follow:

Full conditional posterior of μ

$$p(\mu \mid \sigma^2, X) \sim \mathcal{N}(\mu_n, \tau_n^2)$$

WHERE:

$$\frac{1}{\tau_n^2} = \frac{n}{\sigma^2} + \frac{1}{\tau_0^2}$$

$$\mu_n = w\bar{X} + (1 - w)\mu_0$$

$$w = \frac{\frac{n}{\sigma^2}}{\frac{n}{\sigma^2} + \frac{1}{\tau_0^2}}$$

Full conditional posterior of σ^2

$$p(\sigma^2 \mid \mu, X) \sim Inv - \chi^2(\nu_n, \frac{\nu_0\sigma_0^2 + \sum_{i=1}^{n}(X_i - \mu)^2}{n + \nu_0})$$

WHERE

$$\nu_n = n + \nu_0$$

So in each iteration, we draw from full conditional posterior of σ^2, using a previously generated value of μ; then we put this generated value in the full conditional posterior of μ, drawing a new observation for μ that will be used for the next step of the algorithm. In the first iteration, we assumed $\mu = \mu_0$.

The code to achieve the task is the following:

```
# ------------------------------------------------------------------------------
# Q1 - a)
# ------------------------------------------------------------------------------

#--- Normal Model (Using Gibbs sampler)
set.seed(12345)

#--- Prior parameters
mu_0 <- 22.94
tau_0_sqrd <- 25
nu_0 <- 30
sigma_0_sqrd <- 1

#--- Sigma posterior
rinvchisq <- function(draws, n, mu) {
  tau_sqrd <-( nu_0*sigma_0_sqrd + sum((data$rainfall - mu)^2))/(n+nu_0)
  chi_square <- rchisq(draws,n)
  return( tau_sqrd*(n)/chi_square)
}

#--- Gibbs sampler
iterNO <- 3000
mu <- c(22.94)
sigma_post <- numeric()
for(i in 1:iterNO){
  sigma_post <- c(sigma_post,rinvchisq(1,n,mu[i]))
  tau_n_sqrd <- 1/(n/sigma_post[i] + 1/tau_0_sqrd)
  w <- (n/sigma_post[i]) / ((n/sigma_post[i]) + 1/tau_0_sqrd)
  mu_n <- w*mean(data$rainfall) + (1-w)*mu_0
  mu <- c(mu,rnorm(n = 1,mu_n,tau_n_sqrd))
}

# Eliminate the first observations (burn-in)
# burn_in_period <- 100
# mu <- mu[burn_in_period:length(mu)]
# sigma_post <- sigma_post[burn_in_period:length(sigma_post)]
```

We are now asked to analyse graphically the convergence of the Gibbs sampler. In figure 1 we can observe the representation of the MC chains produced.
The plots show how the values converge almost immediately, with practically no burn-in period. There is quite a lot of variation among the generated values. The observed range for μ goes between 31.5 and 33, while the one of σ^2 between 1450 and 1650.
The values to which the sampler converges are 32.2031 and 1540.4965 (blue lines), for μ and σ^2 respectively. Compared to the sample estimate, equal to 32.2831 and 1547.1025 respectively (red lines), the generated values seem to be slightly biased negatively.

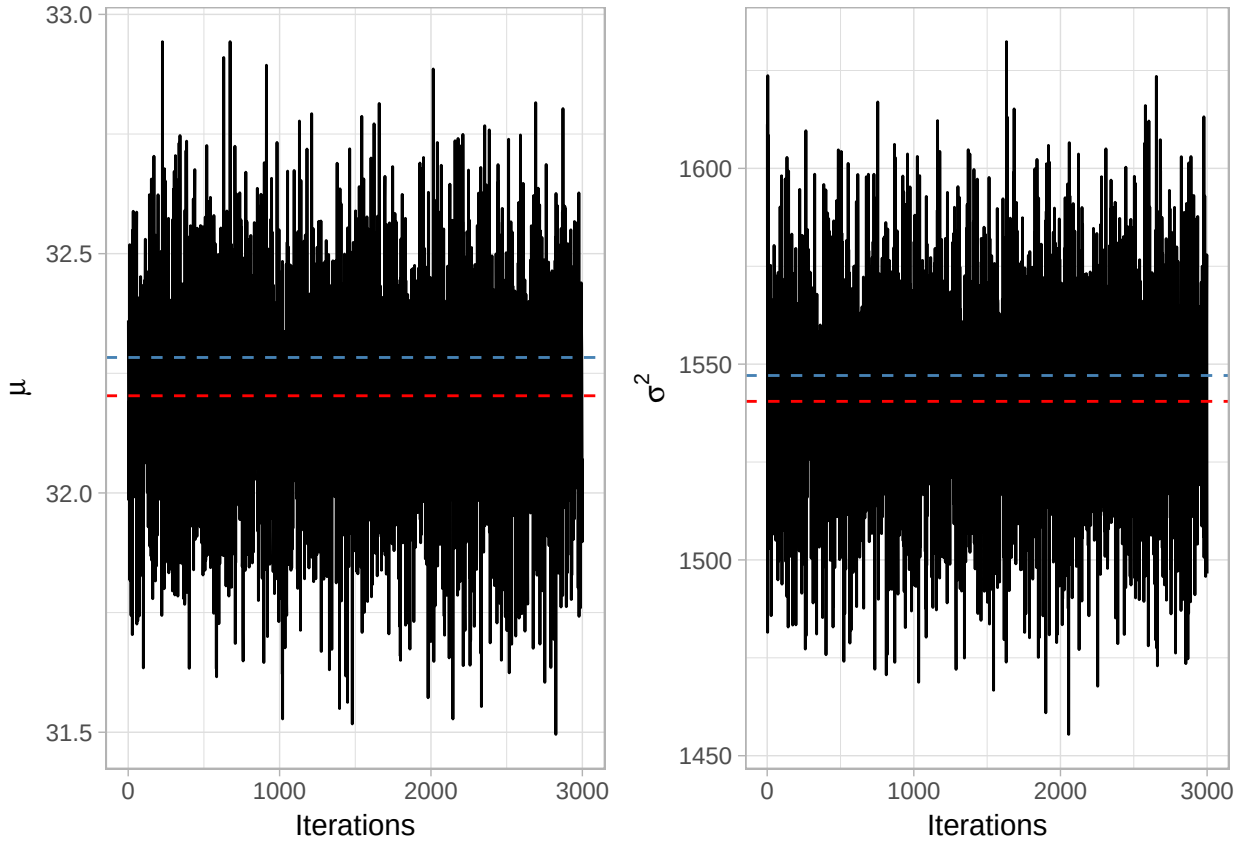

Figure 1: Convergence of the μ and σ^2 parameters using the Gibbs-sampling method. The red dashed line represents the mean of the generated values, while the blue one the estimates from the data.

However, if we observe the real data, the difference in the mean is actually pretty small, since the distribution is heavily skewed on the left. In the greater picture the mean of the data (Figure 2, black dashed line) and the mean of the generated values (red continuous line) practically coincide, suggesting a good accuracy in our generator.
This does not mean however that we will get a good fit: in fact such a random variable does not seem appropriate for the data we posses. Our model underestimates the high density of small values while overestimating the range $[50, 100]$. We will need a combination of different Gaussians or another type of random variable to reach a good fit of our data.

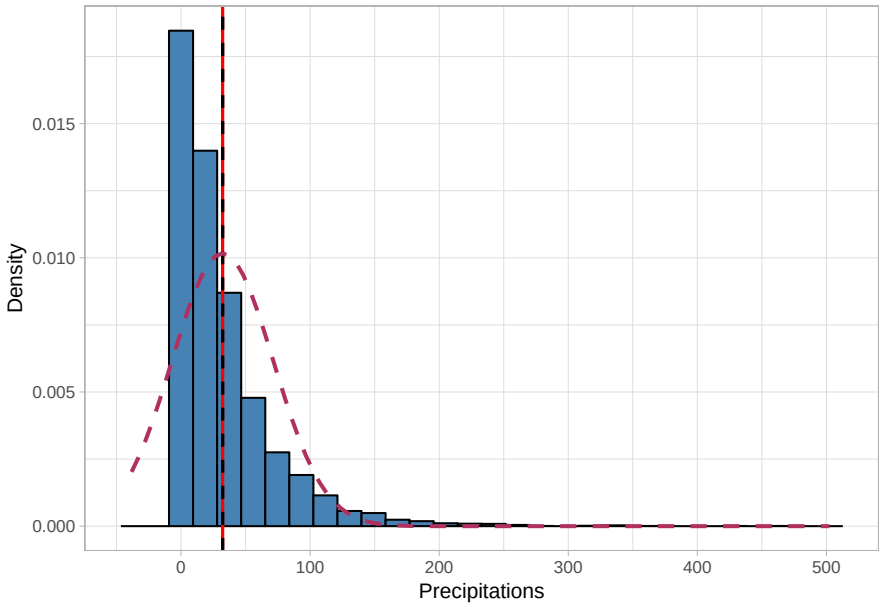

Figure 2: Histogram of the dataset, with the mean of the data (black line) and the mean of the generated values (red line, overlapping), together with the density of the fitted model (purple line).

(b) Mixture of normals model

We are now asked to implement a model assuming the daily precipitation $\{y_1, \dots, y_n\}$ follow an iid two-component mixture of normals model as follows:

$$p(y_i \mid \mu, \sigma^2, \pi) = \pi \mathcal{N}(y_i \mid \mu_1, \sigma_1^2) + (1-\pi)\mathcal{N}(y_i \mid \mu_2, \sigma_2^2) \qquad \text{where:} \quad \begin{cases} \mu = (\mu_1, \mu_2) \\ \sigma^2 = (\sigma_1^2, \sigma_2^2) \end{cases}$$

This type of model leads to the following prior distributions:

$$\pi \sim Beta(\alpha_1, \alpha_2) \qquad \beta_j \mid \sigma_j^2 \sim \mathcal{N}(\mu_{0j}, \sigma_j^2) \qquad \sigma_j^2 \sim Inv - \chi^2(\nu_{0j}, \sigma_{0j}^2)$$

Reusing the code provided during the lectures (`NormalMixtureGibbs.R`) we created a Gibbs sampling to analyse the daily precipitation data.
The most important part is to set the prior parameters of the distribution suitably. From our prior information however it does not seem such a model will make much sense for the phenomena, since it follows a cyclical structure along the months of the year. Moreover looking at the data itself in the previous task it appears to us this kind of model will not be able suitable to the nature of the phenomena: we may expect one component to model the most probable values (close to zero) and then another one to model the outliers. Nonetheless the precipitations are exclusively POSITIVE values, therefore making a component with a really wide variance (such as the one for the outliers) we expect also many negative values to be modelled as well. Probably a *Poisson* or *Exponential* model better fit the data.

Being unsure about what to expect from this model, we decided to set the prior parameters of both components equal to the ones of the previous step, so $\mu_0 = (22.94, 22.94)$, $\sigma_0^2 = (1, 1)$, $\tau_0^2 = (25, 25)$ and $\nu_0^2 = (30, 30)$. We assumed equally likely components as well, setting $\alpha = (10, 10)$.
Our adaptation of the script `NormalMixtureGibbs.R` is the following:

```r
#-----------------------------------------------------
#  Q1 - b)
#-----------------------------------------------------

x <- as.matrix(data)

# Model options
nComp <- 2                                # Number of mixture components

# Prior options
alpha <- 10*rep(1,nComp)                  # Dirichlet(alpha)
muPrior <- rep(mu_0,nComp)                # Prior mean of mu
tau2Prior <- rep(tau_0_sqrd,nComp)        # Prior std of mu
sigma2_0 <- rep(sigma_0_sqrd,nComp)       # s20 (best guess of sigma2)
nu0 <- rep(nu_0,nComp)                    # degrees of freedom for prior on sigma2

# MCMC options
nIter <- 1000                             # Number of Gibbs sampling draws

# Defining a function that simulates from the inverse chisquare
rScaledInvChi2 <- function(n, df, scale){
  return((df*scale)/rchisq(n,df=df))
}

# Defining a function that simulates from a Dirichlet distribution
rDirichlet <- function(param){
  nCat <- length(param)
  piDraws <- matrix(NA,nCat,1)
  for (j in 1:nCat){
    piDraws[j] <- rgamma(1,param[j],1)
  }
  piDraws = piDraws/sum(piDraws) # Diving every column of piDraws by the sum
                                 # of the elements in that column.
  return(piDraws)
}

# Simple function that converts between two different representations
# of the mixture allocation
S2alloc <- function(S){
  n <- dim(S)[1]
  alloc <- rep(0,n)
  for (i in 1:n){
    alloc[i] <- which(S[i,] == 1)
  }
  return(alloc)
}

# Initial value for the MCMC
nObs <- length(x)
# nObs-by-nComp matrix with component allocations.
S <- t(rmultinom(nObs, size = 1 , prob = rep(1/nComp,nComp)))
mus <- quantile(x, probs = seq(0,1,length = nComp))
```

```
sigma2 <- rep(var(x),nComp)
probObsInComp <- rep(NA, nComp)

# Store the values
sigmas_mix <- matrix(NA, nIter, 2)
mu_mix <- matrix(NA, nIter, 2)
pi_mix <- rep(NA, nIter)

for (k in 1:nIter){

  alloc <- S2alloc(S) # Just a function that converts between
                      # different representations of the group allocations
  nAlloc <- colSums(S)

  # Update components probabilities
  pi <- rDirichlet(alpha + nAlloc)
  pi_mix[k] <- pi[1]

  # Update mu's
  for (j in 1:nComp){
    precPrior <- 1/tau2Prior[j]
    precData <- nAlloc[j]/sigma2[j]
    precPost <- precPrior + precData
    wPrior <- precPrior/precPost
    muPost <- wPrior*muPrior + (1-wPrior)*mean(x[alloc == j])
    tau2Post <- 1/precPost
    mus[j] <- rnorm(1, mean = muPost, sd = sqrt(tau2Post))
  }

  mu_mix[k,] <- mus

  # Update sigma2's
  for (j in 1:nComp){
    scale = (nu0[j]*sigma2_0[j] + sum((x[alloc == j] - mus[j])^2)) /
      (nu0[j] + nAlloc[j])
    sigma2[j] <- rScaledInvChi2(1, df = nu0[j] + nAlloc[j], scale = scale)
  }

  sigmas_mix[k,] <- sigma2

  # Update allocation
  for (i in 1:nObs){
    for (j in 1:nComp){
      probObsInComp[j] <- pi[j]*dnorm(x[i], mean = mus[j], sd = sqrt(sigma2[j]))
    }
    S[i,] <- t(rmultinom(1, size = 1 , prob = probObsInComp/sum(probObsInComp)))
  }

}
```

After running the script, we plotted the trend of the generated MCMC chains in order to evaluate once more the convergence of the two pairs of parameters. In Figure 3 we can find the plots for (μ_1, μ_2), while in Figure

4 we can find the plots for (σ_1^2, σ_2^2).
In both parameters each component starts from the sample mean (or variance) of the data and is characterized by a peculiar behaviour: component 1 immediately drops to a low mean and variance, creating a really narrow distribution close to zero; component 2 instead after spiking to huge values it stabilizes around smaller (yet still really high) numbers, generating an drastically flat distribution centred at extreme values.
This time we have a clear and relevant burn-in period of approximately 100 observations. The blue convergence lines include all the generated values while the red ones exclude the burn-in: we can observe relevant differences between them, especially for μ_1 and σ_2^2. Nevertheless in both cases all the chains converge to a stable value, even though for μ the values appear to be more correlated than in the other pair of parameters.

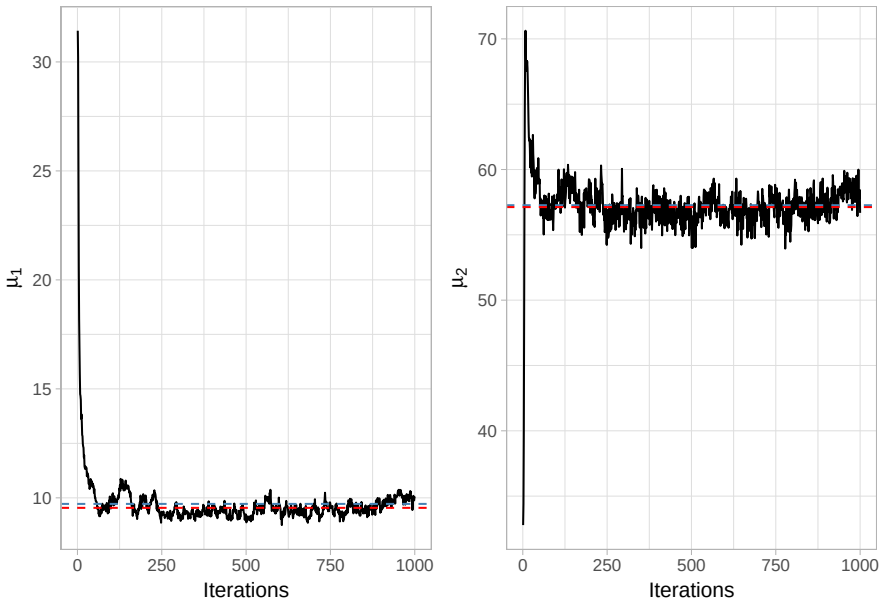

Figure 3: MCMC plots for the means (μ_1, μ_2) of the two components.

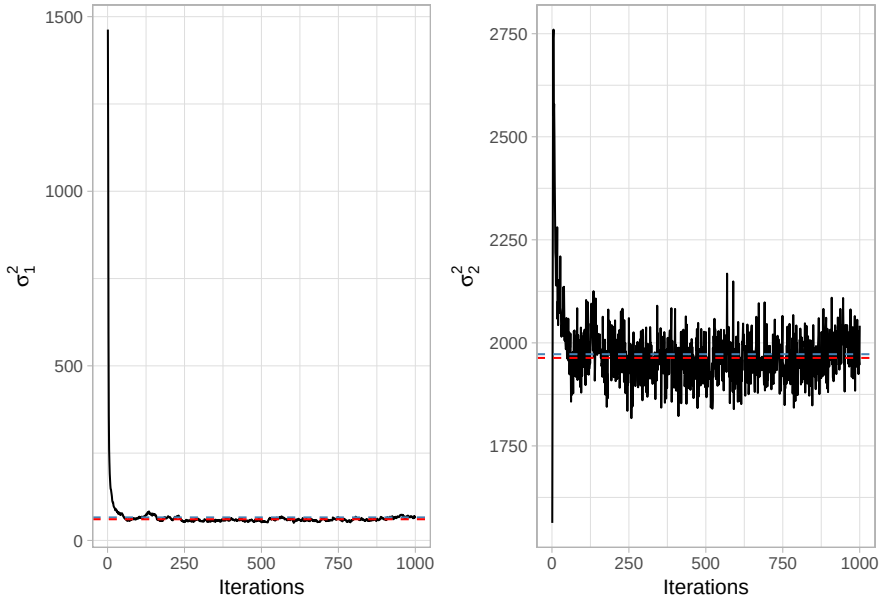

Figure 4: MCMC plots for the variances (σ_1^2, σ_2^2) of the two components.

The interpretation we can give to this mixed model is essentially the one we hypothesized previously: one of the components (component 1) models the most frequent values, corresponding to low precipitations; the other instead (component 2) follows the extreme values and tries to explain them with a really flat normal distribution. The estimates we get of the two pairs of parameters (excluding the burn-in period) are in fact: $\hat{\mu} = (9.5403, 57.1246)$ and $\hat{\sigma}^2 = (60.6261, 1963.4577)$.
The weight given to each component is almost the same: the probability $\hat{\pi}$ of generating values from component 1 is in fact 0.5306, meaning that we will expect our mixture to be composed of an almost equal proportion of the two components. The MCMC for this parameter (figure 5) follows the same behaviour of the previously observed component 2 and reaches convergence after a burn-in of 100 observations as well. Also in this case the values appear to be highly correlated.

If we observe the plot of the fitted distribution over the dataset (figure 6) we see how poorly the model performed: the mixture of the two distribution leads to overestimate the density of values around $\hat{\mu}_1 = 9.5403$ while underestimating the density of the values immediately after; the second component manages to mitigate this behaviour but it does not fit the data that well either.
Even though the type of model (mixture of two Gaussian) may not be the best for this type of problem, choosing other prior parameters did not change much the outcome. We tried with a much higher degree of belief in our priors $\nu_0 = (150, 150)$, we changed the prior parameters of the model to already reflect the outcome we got and we tried to put an extremely unbalanced probability π with $\alpha = (150, 15)$: neither of this changes nor any combination of them lead to consistent differences.

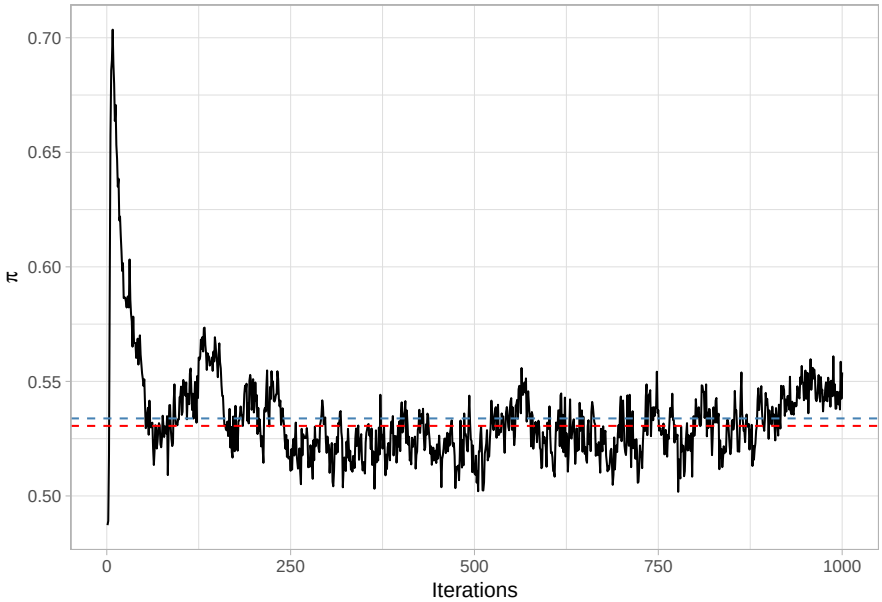

Figure 5: MCMC plots for the probability π of observing component 1.

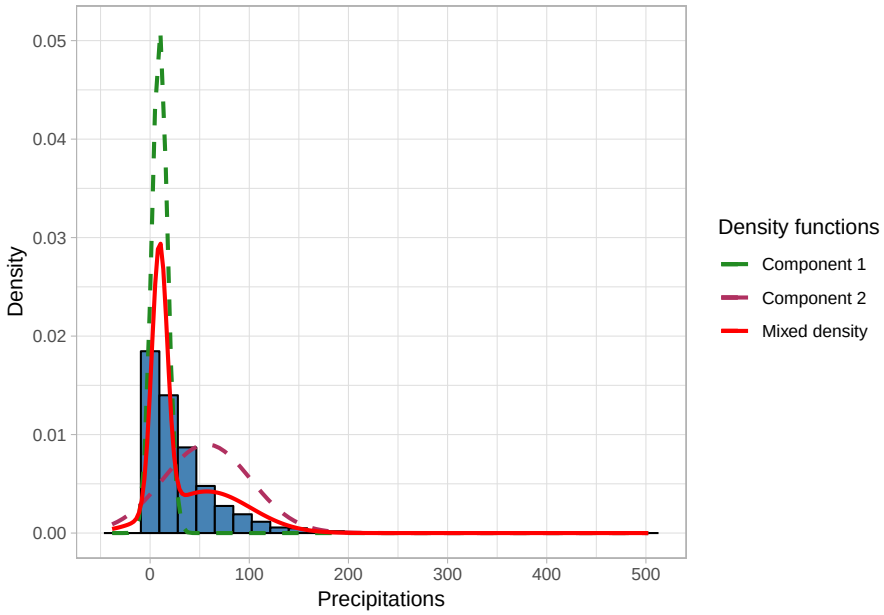

Figure 6: Fitted mixed density and individual component densities on the ‘rainfall‘ dataset.

(c) Graphical comparison

We are now asked to compare the two models fitted in (a) and (b) with a kernel density estimation of the data. In figure 7 we see the three density together: of the two models, the one who gets closer to the kernel density is definitely the mixture of the two Gaussian but, as previously stated, still does not model the trend of the data fully. Most likely an *Exponential*/*Poisson* model will fix the issue and get a better estimate of the density.

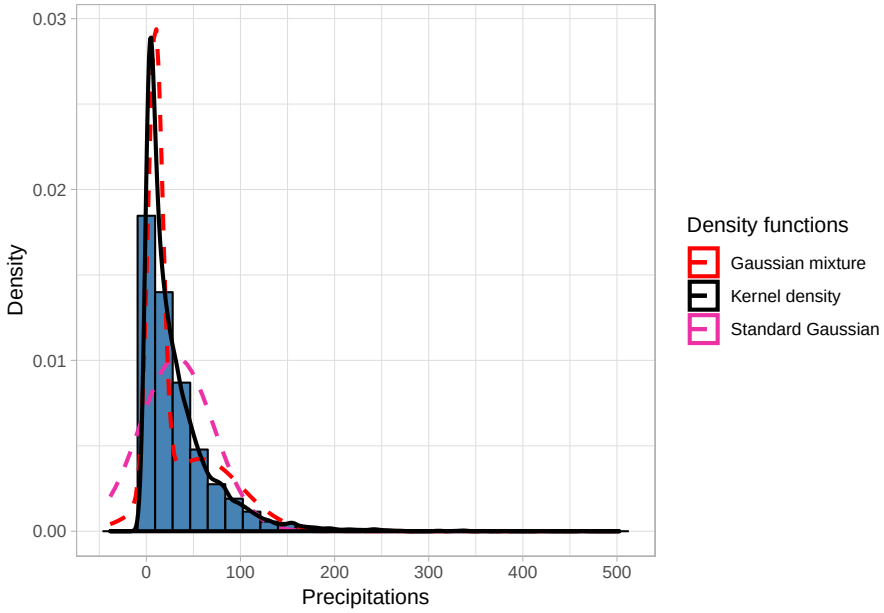

Figure 7: Graphical comparison of the two fitted model with the estimated kernel density.

Question 2 - Metropolis Random Walk for Poisson regression

In this task we are asked to analyse the dataset `eBayNumberOfBidderData.dat`. The file is about the auctions of 100 coins on eBay. We are interest in modelling the response variable **nBids**, representing the number of bids in each auction. We are given the following set of regressors:

- **Const** (for the intercept)
- **PowerSeller** (is the seller selling large volumes on eBay?)
- **VerifyID** (is the seller verified by eBay?)
- **Sealed** (was the coin sold sealed in never opened envelope?)
- **MinBlem** (did the coin have a minor defect?)
- **MajBlem** (a major defect?)
- **LargNeg** (did the seller get a lot of negative feedback from customers?)
- **LogBook** (logarithm of the coins book value according to expert sellers. Standardized)
- **MinBidShare** (a variable that measures ratio of the minimum selling price (starting price) to the book value. Standardized)

In all the successive steps we will try to fit to the dataset a *Poisson* model in the form:

$$y_i \mid \beta \sim \text{Poisson}[\exp(x_i^T \beta)],\ i = 1, \dots, n$$

where y_i is the count for the i^{th} observation in the sample and x_i is the p-dimensional vector with covariate observations for the i^{th} observation.

(a) MLM from GLM

In this first step we are asked to fit the model through the *exponential family*. Fitting a GLM model using the `glm(·)` function from `R`. In Figure 8 are summarized the maximum likelihood estimator of β.
As we can observe, only 3 of the 9 variables result not-significant (`LargNeg`, `MinBlem` and `PowerSeller`). In other words, bidders do not seem discouraged by a large number of negative feedback over the seller and minor damages on the products are almost ignored by the bidders. Even major damages, despite being significant, show a greater p-value than the rest of the significant variable (0.015 approx). Of all the variables considered `MinBidShare`, the one expressing how much overpriced/under prized the coin was, is the one influencing the most the number of bids.

```
fit_glm <- glm(nBids ~ 0 + ., data = data, family = poisson)
```

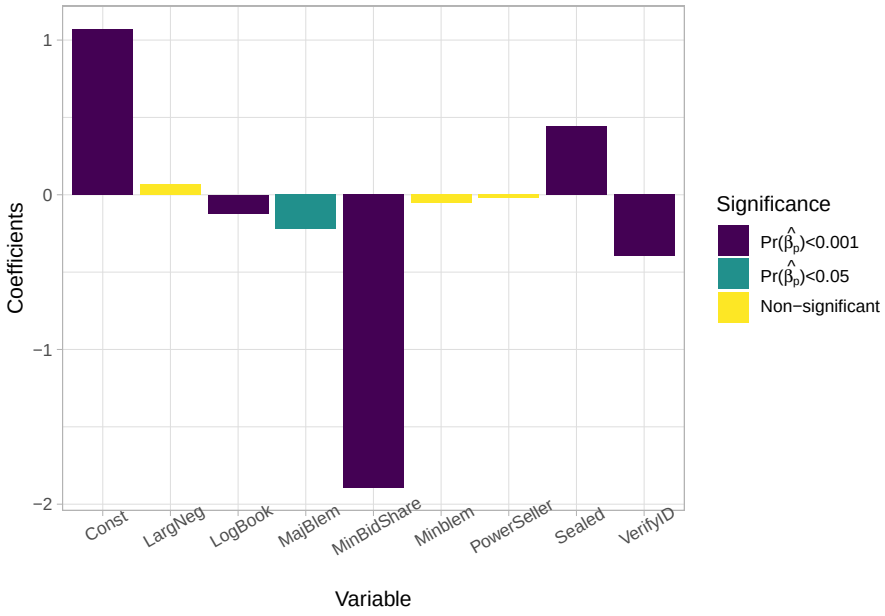

Figure 8: Barplot of the estimated coefficients using GLM.

(b) Bayesian analysis for posterior Poisson

In order to run a Bayesian analysis over the dataset we will set a *Zellner's g-prior* for our coefficients β. The followings are the formulas for the priors and the procedure to compute the posterior:

Zellner's g-prior:

$$\beta \sim \mathcal{N}(0, 100 * (X^T X)^{-1})$$

Likelihood:

$$y_i \mid \beta \sim Poisson[\exp(X_i^T \beta)]$$

$$Likelihood = \prod_{i=1}^{n} p(y_i \mid \beta) = \prod_{i=1}^{n} \frac{e^{y_i (X_i^T \beta) - \exp(X_i^T \beta)}}{y_i!}$$

$$logLikelihood = \sum_{i=1}^{n} (y_i (X_i^T \beta) - \exp(X_i^T \beta)) - \sum_{i=1}^{n} log(y_i!)$$

Posterior density function:

$$logPosterior = logPrior + logLikelihood$$

However, no closed form of this density is available, so we will need to figure out an ulterior step in order to compute it. In this task we will make a multivariate normal approximation of the distribution. As we did in Lab2, we will assume our posterior to be:

$$\beta|y,X \sim \mathcal{N}\left(\tilde{\beta}, J_y^{-1}(\tilde{\beta})\right) \quad \text{where} : \begin{cases} \tilde{\beta} & \text{is the posterior mode} \\ J(\tilde{\beta}) = -\frac{\partial^2 \ln(p(\beta|y))}{\partial\beta\partial\beta^T}\Big|_{\beta=\tilde{\beta}} & \text{is the observed Hessian evaluated at } \tilde{\beta} \end{cases}$$

Likewise, we will compute the posterior mode and Hessian through numeric optimization using `optim(·)` from `R`. We adapted our code from Lab2 as follows:

```r
library(mvtnorm)

# Set the parameters of the prior distribution (Zellner's g-prior)
data <- as.matrix(data)
X <- data[,-1]
Y <- data[, 1]
mu_0 <- rep(0, p)
sigma_0 <- 100 * solve(t(X)%*%X)
mu_start <- mu_0

# Create a function to compute the posterior distribution
poisson_post <- function(starting_beta, mu_prior, var_prior, x, y) {

  # log(lambda) and likelihood computation
  # log_lambda <- apply(x, 1, function(row) sum(row*starting_beta))
  log_lambda <- x%*%starting_beta
  loglik <- sum(y*log_lambda - exp(log_lambda) - log(factorial(y)))
  # Compute the density according to the normal prior
  logprior <- dmvnorm(starting_beta, mu_prior, var_prior, log = T)

  return( loglik + logprior )

}

# Numerical optimization
optimal_res <- optim(mu_start, poisson_post, gr = NULL, mu_0, sigma_0, X, Y,
                     method = "BFGS", control = list(fnscale = -1), hessian = T)

# Extract the results
post_beta_mode <- optimal_res$par
post_cov <- solve(-optimal_res$hessian)
colnames(post_cov) <- colnames(X)
rownames(post_cov) <- colnames(X)
approx_post_sd <- sqrt(diag(post_cov))
```

We then wanted to compare the two methods used to estimate the coefficients. Table 1 consists of a summary of the various estimates for the two methods and their differences. As we can see, the two methods return almost exactly the same values, suggesting that our normal approximation of the Bayesian estimates is really accurate and does not distort much the estimation process.

Table 1: Comparison between GLM and Bayesian normal approximation for the estimation of the β coefficients.

	Const	PowerSeller	VerifyID	Sealed	Minblem	MajBlem	LargNeg	LogBook	MinBidShare
GLM	1.0724421	-0.0205408	-0.3945165	0.4438426	-0.0521983	-0.2208712	0.0706725	-0.1206776	-1.8940966
Bayesian	1.0698412	-0.0205125	-0.3930060	0.4435555	-0.0524663	-0.2212384	0.0706968	-0.1202177	-1.8919850
Difference	0.0026009	-0.0000283	-0.0015105	0.0002871	0.0002680	0.0003672	-0.0000244	-0.0004599	-0.0021116

(c) Metropolis algorithm

In this part we are suppose to create a general function for implementing Metropolis algorithm, such that it could work on different log-posterior functions. For this reason, we will rename the parameters as a more general θ. We will therefore end up with this:

$$\theta_p|\theta^{(i-1)} \sim \mathcal{N}\left(\theta^{(i-1)}, c \cdot \Sigma\right) \quad \text{where}: \begin{cases} \Sigma = J_y^{-1}(\tilde{\beta}) \ \text{ obtained in b) step} \\ c \ \text{ is a tuning parameter, input of the Metropolis function} \end{cases}$$

We use log-posterior densities (instead of just the densities), since logs are more stable and avoids problems with too small or large numbers (overflow). By using log-density, the ratio of posteriors for α calculation as a step in Metropolis algorithm will be as follow:

$$\frac{p(\theta_p \mid y)}{p(\theta^{(i-1)} \mid y)} = \exp[\ln(p(\theta_p \mid y)) - \ln(p(\theta^{(i-1)} \mid y))]$$

The code to achieve the Metropolis Function is the following:

```
#-----------------------------------------------------------------
# (c) Metropolis Function
#-----------------------------------------------------------------

RWMSampler <- function(startingPoint,nIter,c,logPostFunc,...){

  covarianceMatrix <- post_cov
  drawn_points <- matrix(0,ncol = length(startingPoint), nrow = (nIter+1))
  drawn_points[1,] <- startingPoint

  for(i in 1:nIter){
    drawn_points[(i+1),] <- rmvnorm(n = 1, mean = drawn_points[i,]
                                    ,sigma = c*as.matrix(covarianceMatrix))

    logPost_ratio <- exp(logPostFunc(drawn_points[(i+1),],...)
                         - logPostFunc(drawn_points[i,],...))

    alpha <- min(1,logPost_ratio)
    if(alpha < runif(1,0,1)){
      drawn_points[(i+1),] <- drawn_points[i,]
    }

  }
  drawn_points
}
```

```
#---------- using of Metropolis function to sample from the posterior

metropolisPoints <- RWMSampler(startingPoint = mu_start,nIter = 5000
                    ,c = 0.65,logPostFunc = poisson_post, mu_0, sigma_0, X, Y)
```

In order to asses the MCMC convergence of our functions, we plotted the various chains for each coefficient β_j. The resulting 9 plots are displayed in Figure 9: they all manage to reach convergence in within 1000 observation (the burn-in period) and there are no visible flat points, suggesting that there are no considerable rejection sequences. Our generation process seems therefore trustable.

We finally compare the Metropolis Hasting algorithm with the normal approximation used in step b). Table 2 consists of a summary of the various estimates for the two methods and their differences. As we can see, again the differences are extremely small, meaning that both the Metropolis algorithm and the normal approximation lead to trustable results. Since the difference is so small (both in absolute and relative terms) one may decide to use the normal approximation in order to save computational time and resources, since the Metropolis function is much more time-consuming.

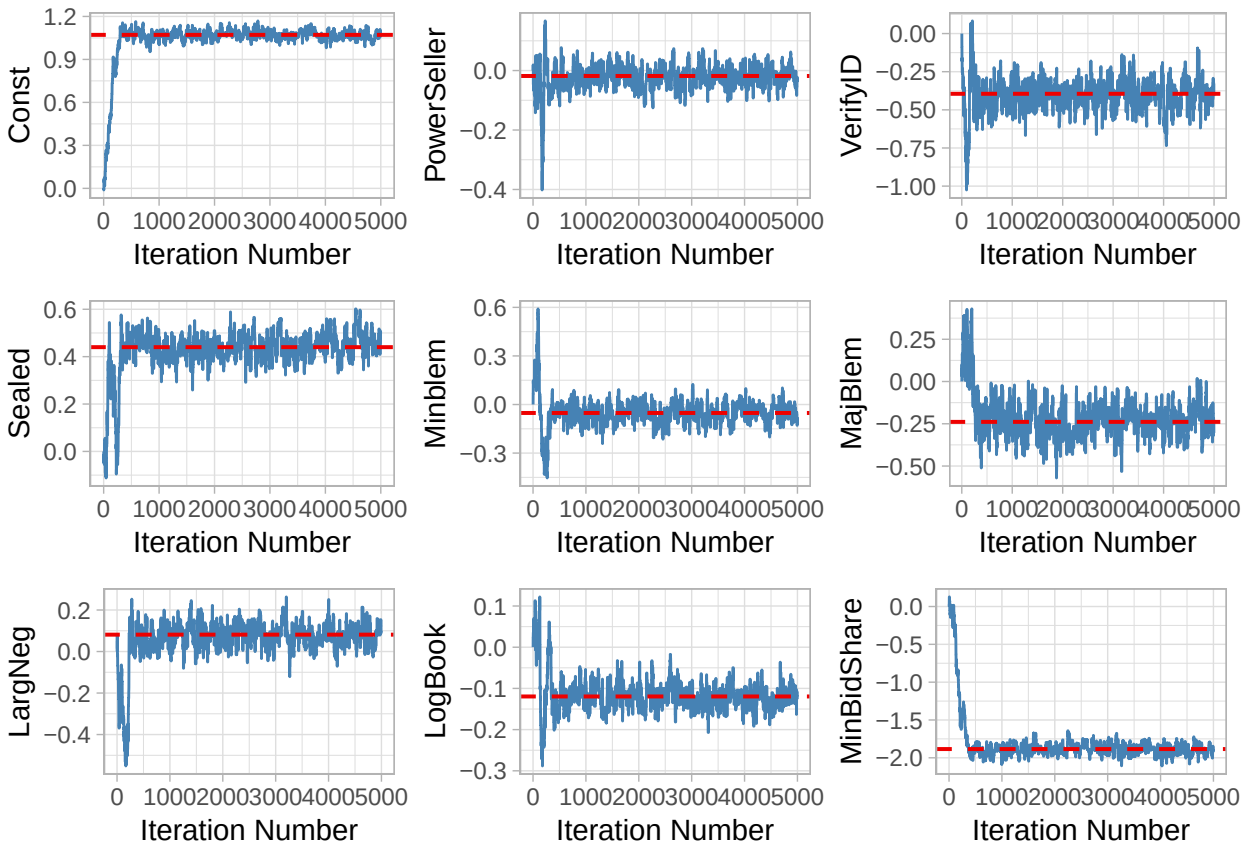

Figure 9: MCMC chains of all the 9 β coefficients and related converged value (red line, burn-in not considered).

Table 2: Comparison between Bayesian normal approximation and Metropolis Hasting algorithm for the estimation of the β coefficients.

	Const	PowerSeller	VerifyID	Sealed	Minblem	MajBlem	LargNeg	LogBook	MinBidShare
Approximation	1.0698412	-0.0205125	-0.393006	0.4435555	-0.0524663	-0.2212384	0.0706968	-0.1202177	-1.891985
Metropolis	1.0713409	-0.0182730	-0.394856	0.4400905	-0.0522395	-0.2388311	0.0812317	-0.1195474	-1.884478
Difference	-0.0014997	-0.0022394	0.001850	0.0034650	-0.0002268	0.0175927	-0.0105349	-0.0006703	-0.007507

Appendix

```
knitr::opts_chunk$set(echo = F, message = F, error = F, warning = F,
                      fig.align='center', out.width="70%")


# ------------------------------------------------------------------------------
# Q1 - Intro
# ------------------------------------------------------------------------------

# Read the data
data <- read.delim("rainfall.dat", header = FALSE)
colnames(data) <- "rainfall"
n <- nrow(data)
p <- ncol(data)


# ------------------------------------------------------------------------------
# Q1 - a)
# ------------------------------------------------------------------------------

#--- Normal Model (Using Gibbs sampler)
set.seed(12345)

#--- Prior parameters
mu_0 <- 22.94
tau_0_sqrd <- 25
nu_0 <- 30
sigma_0_sqrd <- 1

#--- Sigma posterior
rinvchisq <- function(draws, n, mu) {
  tau_sqrd <-( nu_0*sigma_0_sqrd + sum((data$rainfall - mu)^2))/(n+nu_0)
  chi_square <- rchisq(draws,n)
  return( tau_sqrd*(n)/chi_square)
}

#--- Gibbs sampler
iterNO <- 3000
mu <- c(22.94)
sigma_post <- numeric()
for(i in 1:iterNO){
  sigma_post <- c(sigma_post,rinvchisq(1,n,mu[i]))
  tau_n_sqrd <- 1/(n/sigma_post[i] + 1/tau_0_sqrd)
  w <- (n/sigma_post[i]) / ((n/sigma_post[i]) + 1/tau_0_sqrd)
```

```r
  mu_n <- w*mean(data$rainfall) + (1-w)*mu_0
  mu <- c(mu,rnorm(n = 1,mu_n,tau_n_sqrd))
}

# Eliminate the first observations (burn-in)
# burn_in_period <- 100
# mu <- mu[burn_in_period:length(mu)]
# sigma_post <- sigma_post[burn_in_period:length(sigma_post)]


# Plot for parameter convergence
library(ggplot2)
df_plt <- data.frame(mu = mu[-1], sigma = sigma_post,
                     # sample_size = burn_in_period:iterNO)
                     sample_size = 1:iterNO)

p1 <- ggplot(df_plt, aes(x = sample_size)) +
  geom_line(aes(y = mu)) +
  geom_hline(aes(yintercept = mean(mu[-1])), col = "red", lty = 2) +
  geom_hline(aes(yintercept = mean(data$rainfall)), col = "steelblue", lty = 2) +
  labs(y = expression(mu), x = "Iterations") +
  theme_light()

p2 <- ggplot(df_plt, aes(x = sample_size)) +
  geom_line(aes(y = sigma)) +
  geom_hline(aes(yintercept = mean(sigma)), col = "red", lty = 2) +
  geom_hline(aes(yintercept = var(data$rainfall)), col = "steelblue", lty = 2) +
  labs(y = expression(sigma^2), x = "Iterations") +
  theme_light()

gridExtra::grid.arrange(p1, p2, ncol = 2)


# Setting up the plot
x <- as.matrix(data)
xGrid <- seq(min(x)-1*apply(x,2,sd), max(x)+1*apply(x,2,sd),length = 200)

# Histogram of the data
ggplot() +
  geom_histogram(aes(x = rainfall, y = ..density..), data = data,
                 fill = "steelblue", col = "black") +
  geom_vline(aes(xintercept = mean(mu[-1])), lty = 1, size = .8, col = "red") +
  geom_vline(aes(xintercept = mean(data$r)), lty = 2, size = .8, col = "black") +
  stat_function(aes(x = xGrid), fun = dnorm, size = 1, lty = 2, col = "maroon",
                args = list(mean = mean(mu[-1]), sd = sqrt(mean(sigma_post)))) +
  labs(y = "Density", x = "Precipitations") +
  theme_light()


#-------------------------------------------------------
#  Q1 - b)
#-------------------------------------------------------
```

```r
x <- as.matrix(data)

# Model options
nComp <- 2                                  # Number of mixture components

# Prior options
alpha <- 10*rep(1,nComp)                    # Dirichlet(alpha)
muPrior <- rep(mu_0,nComp)                  # Prior mean of mu
tau2Prior <- rep(tau_0_sqrd,nComp)          # Prior std of mu
sigma2_0 <- rep(sigma_0_sqrd,nComp)         # s20 (best guess of sigma2)
nu0 <- rep(nu_0,nComp)                      # degrees of freedom for prior on sigma2

# MCMC options
nIter <- 1000                               # Number of Gibbs sampling draws

# Defining a function that simulates from the inverse chisquare
rScaledInvChi2 <- function(n, df, scale){
  return((df*scale)/rchisq(n,df=df))
}

# Defining a function that simulates from a Dirichlet distribution
rDirichlet <- function(param){
  nCat <- length(param)
  piDraws <- matrix(NA,nCat,1)
  for (j in 1:nCat){
    piDraws[j] <- rgamma(1,param[j],1)
  }
  piDraws = piDraws/sum(piDraws) # Diving every column of piDraws by the sum
                                 # of the elements in that column.
  return(piDraws)
}

# Simple function that converts between two different representations
# of the mixture allocation
S2alloc <- function(S){
  n <- dim(S)[1]
  alloc <- rep(0,n)
  for (i in 1:n){
    alloc[i] <- which(S[i,] == 1)
  }
  return(alloc)
}

# Initial value for the MCMC
nObs <- length(x)
# nObs-by-nComp matrix with component allocations.
S <- t(rmultinom(nObs, size = 1 , prob = rep(1/nComp,nComp)))
mus <- quantile(x, probs = seq(0,1,length = nComp))
sigma2 <- rep(var(x),nComp)
probObsInComp <- rep(NA, nComp)

# Store the values
sigmas_mix <- matrix(NA, nIter, 2)
```

```r
mu_mix <- matrix(NA, nIter, 2)
pi_mix <- rep(NA, nIter)

for (k in 1:nIter){

  alloc <- S2alloc(S) # Just a function that converts between
                      # different representations of the group allocations
  nAlloc <- colSums(S)

  # Update components probabilities
  pi <- rDirichlet(alpha + nAlloc)
  pi_mix[k] <- pi[1]

  # Update mu's
  for (j in 1:nComp){
    precPrior <- 1/tau2Prior[j]
    precData <- nAlloc[j]/sigma2[j]
    precPost <- precPrior + precData
    wPrior <- precPrior/precPost
    muPost <- wPrior*muPrior + (1-wPrior)*mean(x[alloc == j])
    tau2Post <- 1/precPost
    mus[j] <- rnorm(1, mean = muPost, sd = sqrt(tau2Post))
  }

  mu_mix[k,] <- mus

  # Update sigma2's
  for (j in 1:nComp){
    scale = (nu0[j]*sigma2_0[j] + sum((x[alloc == j] - mus[j])^2)) /
      (nu0[j] + nAlloc[j])
    sigma2[j] <- rScaledInvChi2(1, df = nu0[j] + nAlloc[j], scale = scale)
  }

  sigmas_mix[k,] <- sigma2

  # Update allocation
  for (i in 1:nObs){
    for (j in 1:nComp){
      probObsInComp[j] <- pi[j]*dnorm(x[i], mean = mus[j], sd = sqrt(sigma2[j]))
    }
    S[i,] <- t(rmultinom(1, size = 1 , prob = probObsInComp/sum(probObsInComp)))
  }

}


# Plot for mus
burn <- 100
df_plt <- data.frame(mu_mix, sample_size = 1:k)
colnames(df_plt) <- c("mu1", "mu2", "sample_size")

p1 <- ggplot(df_plt, aes(x = sample_size)) +
  geom_line(aes(y = mu1)) +
```

```r
  geom_hline(aes(yintercept = mean(mu1)), col = "steelblue", lty = 2) +
  geom_hline(aes(yintercept = mean(mu1[-c(1:burn)])), col = "red", lty = 2) +
  labs(y = expression(mu[1]), x = "Iterations") +
  theme_light()

p2 <- ggplot(df_plt, aes(x = sample_size)) +
  geom_line(aes(y = mu2)) +
  geom_hline(aes(yintercept = mean(mu2)), col = "steelblue", lty = 2) +
  geom_hline(aes(yintercept = mean(mu2[-c(1:burn)])), col = "red", lty = 2) +
  labs(y = expression(mu[2]), x = "Iterations") +
  theme_light()

gridExtra::grid.arrange(p1, p2, ncol = 2)


# Plot for sigmas
burn <- 100
df_plt <- data.frame(sigmas_mix, sample_size = 1:k)
colnames(df_plt) <- c("sigma1", "sigma2", "sample_size")

p1 <- ggplot(df_plt, aes(x = sample_size)) +
  geom_line(aes(y = sigma1)) +
  geom_hline(aes(yintercept = mean(sigma1)), col = "steelblue", lty = 2) +
  geom_hline(aes(yintercept = mean(sigma1[-c(1:burn)])), col = "red", lty = 2) +
  labs(y = expression(sigma[1]^2), x = "Iterations") +
  theme_light()

p2 <- ggplot(df_plt, aes(x = sample_size)) +
  geom_line(aes(y = sigma2)) +
  geom_hline(aes(yintercept = mean(sigma2)), col = "steelblue", lty = 2) +
  geom_hline(aes(yintercept = mean(sigma2[-c(1:burn)])), col = "red", lty = 2) +
  labs(y = expression(sigma[2]^2), x = "Iterations") +
  theme_light()

gridExtra::grid.arrange(p1, p2, ncol = 2)


# Plot for pi
burn <- 100
df_plt <- data.frame(pi_mix, sample_size = 1:k)
colnames(df_plt) <- c("pi", "sample_size")

ggplot(df_plt, aes(x = sample_size)) +
  geom_line(aes(y = pi)) +
  geom_hline(aes(yintercept = mean(pi)), col = "steelblue", lty = 2) +
  geom_hline(aes(yintercept = mean(pi[-c(1:burn)])), col = "red", lty = 2) +
  labs(y = expression(pi), x = "Iterations") +
  theme_light()


# Plot for the mixture of components

# Getting the estimates of the parameters
```

```r
burn <- 100
mus_post <- colMeans(mu_mix[-c(1:burn),])
sigmas_post <- colMeans(sigmas_mix[-c(1:burn),])
pi_post <- mean(pi_mix[-c(1:burn)])

# Setting up the plot
xGrid <- seq(min(x)-1*apply(x,2,sd), max(x)+1*apply(x,2,sd),length = 200)
xGridMin <- min(xGrid)
xGridMax <- max(xGrid)
mixDens <- rep(0, length(xGrid))
ylim <- c(0, 2*max(hist(x, plot = F)$density))

# Compute the mixed density
for(i in 1:length(xGrid)) {
  mixDens[i] <- pi_post*dnorm(xGrid[i], mus_post[1], sqrt(sigmas_post[1])) +
    (1-pi_post)*dnorm(xGrid[i], mus_post[2], sqrt(sigmas_post[2]))
}


ggplot() +
  geom_histogram(aes(x = rainfall, y = ..density..), data = data,
                 fill = "steelblue", col = "black") +
  stat_function(aes(x = xGrid, col = "Component 1"), fun = dnorm, size = 1, lty = 2,
                args = list(mean = mus_post[1], sd = sqrt(sigmas_post[1]))) +
  stat_function(aes(x = xGrid, col = "Component 2"), fun = dnorm, size = 1, lty = 2,
                args = list(mean = mus_post[2], sd = sqrt(sigmas_post[2]))) +
  geom_line(aes(x = xGrid, y = mixDens, col = "Mixed density"), size = 1) +
  scale_color_manual(values = c("forestgreen", "maroon", "red"),
                     name = "Density functions") +
  labs(y = "Density", x = "Precipitations") +
  theme_light()


#-------------------------------------------------------------------------------
# (c) Graphical comparison
#-------------------------------------------------------------------------------

ggplot() +
  geom_histogram(aes(x = rainfall, y = ..density..), data = data,
                 fill = "steelblue", col = "black") +
  stat_function(aes(x = xGrid, col = "Standard Gaussian"), fun = dnorm, size = 1, lty = 2,
                args = list(mean = mean(mu[-1]), sd = sqrt(mean(sigma_post)))) +
  geom_line(aes(x = xGrid, y = mixDens, col = "Gaussian mixture"), size = 1, lty = 2) +
  geom_density(aes(x = rainfall, y = ..density.., col = "Kernel density"),
               data = data, size = 1) +
  scale_color_manual(values = c("red", "black", "maroon2"),
                     name = "Density functions") +
  labs(y = "Density", x = "Precipitations") +
  theme_light()


# ------------------------------------------------------------------------------
# Q2 - INTRO
```

```r
# ------------------------------------------------------------------------------

# Read the data
data <- read.delim("eBayNumberOfBidderData.dat",sep = '')
n <- nrow(data)
p <- ncol(data)-1


fit_glm <- glm(nBids ~ 0 + ., data = data, family = poisson)


# For labels in the legend
library(latex2exp)

s <- as.data.frame(summary(fit_glm)$coef)
features_names <- rownames(s)
pval <- s$`Pr(>|z|)`
df_coef <- data.frame(Variable = features_names, Coefficients = s$Estimate,
                      Significance = as.factor(
                        ifelse(pval>0.05, "Non-significant",
                               ifelse(pval>0.01, "alpha = 0.05",
                                      "alpha = 0.001"))))

ggplot(df_coef, aes(x = Variable, y = Coefficients, fill = Significance)) +
  geom_bar(stat = "identity") +
  scale_fill_manual(values = viridis::viridis(3),
                    labels = list(TeX("$\\Pr(\\hat{\\beta_p})<0.001$"),
                                  TeX("$\\Pr(\\hat{\\beta_p})<0.05$"),
                                  TeX("Non-significant"))) +
  theme_light() +
  theme(axis.text.x = element_text(angle = 30))


library(mvtnorm)

# Set the parameters of the prior distribution (Zellner's g-prior)
data <- as.matrix(data)
X <- data[,-1]
Y <- data[, 1]
mu_0 <- rep(0, p)
sigma_0 <- 100 * solve(t(X)%*%X)
mu_start <- mu_0

# Create a function to compute the posterior distribution
poisson_post <- function(starting_beta, mu_prior, var_prior, x, y) {

  # log(lambda) and likelihood computation
  # log_lambda <- apply(x, 1, function(row) sum(row*starting_beta))
  log_lambda <- x%*%starting_beta
  loglik <- sum(y*log_lambda - exp(log_lambda) - log(factorial(y)))
  # Compute the density according to the normal prior
  logprior <- dmvnorm(starting_beta, mu_prior, var_prior, log = T)
```

```
  return( loglik + logprior )

}

# Numerical optimization
optimal_res <- optim(mu_start, poisson_post, gr = NULL, mu_0, sigma_0, X, Y,
                     method = "BFGS", control = list(fnscale = -1), hessian = T)

# Extract the results
post_beta_mode <- optimal_res$par
post_cov <- solve(-optimal_res$hessian)
colnames(post_cov) <- colnames(X)
rownames(post_cov) <- colnames(X)
approx_post_sd <- sqrt(diag(post_cov))


# Create a table showing the differences between the various estimations
df_table <- data.frame(GLM = fit_glm$coefficients, Bayesian = post_beta_mode,
                       Difference = fit_glm$coefficients-post_beta_mode)
df_table <- t(df_table)
names_table <- colnames(X)

library(kableExtra)
kable(df_table, "latex", booktabs = T, align = "c", col.names = names_table,
      linesep = "", escape = F,
      caption = "Comparison between GLM and Bayesian normal approximation
      for the estimation of the $\\beta$ coefficients.") %>%
  column_spec(c(1), border_right = T) %>%
  row_spec(0, bold = T) %>%
  row_spec(2, hline_after = T) %>%
  kable_styling(latex_options = "hold_position", font_size = 7)


#------------------------------------------------------------------
# (c) Metropolis Function
#------------------------------------------------------------------

RWMSampler <- function(startingPoint,nIter,c,logPostFunc,...){

  covarianceMatrix <- post_cov
  drawn_points <- matrix(0,ncol = length(startingPoint), nrow = (nIter+1))
  drawn_points[1,] <- startingPoint

  for(i in 1:nIter){
    drawn_points[(i+1),] <- rmvnorm(n = 1, mean = drawn_points[i,]
                                    ,sigma = c*as.matrix(covarianceMatrix))

    logPost_ratio <- exp(logPostFunc(drawn_points[(i+1),],...)
                         - logPostFunc(drawn_points[i,],...))

    alpha <- min(1,logPost_ratio)
    if(alpha < runif(1,0,1)){
      drawn_points[(i+1),] <- drawn_points[i,]
```

```
    }

  }
  drawn_points
}

#---------- using of Metropolis function to sample from the posterior

metropolisPoints <- RWMSampler(startingPoint = mu_start,nIter = 5000
                    ,c = 0.65,logPostFunc = poisson_post, mu_0, sigma_0, X, Y)


#----------- Assessing MCMC convergence by graphical methods
metropolisPoints <- as.data.frame(metropolisPoints)
metropolisPoints <- cbind(c(1:nrow(metropolisPoints)),metropolisPoints)
colnames(metropolisPoints) <- c('nIter',colnames(data)[-1])

plot_fun <- function(col, burn = 1000) {
  ggplot(data = metropolisPoints, mapping = aes(x = c(1:nrow(metropolisPoints)))) +
  geom_line(mapping = aes(y = metropolisPoints[,col]), col = "steelblue") +
  geom_hline(yintercept = mean(metropolisPoints[-c(1:burn),col]),
             size = .7, lty = 2, col = "red2") +
  labs(y = col, x = "Iteration Number") +
  theme_light()
}


names <- colnames(metropolisPoints)[-1]

grbs <- lapply(X = names, FUN = plot_fun)

library(gridExtra)
grid.arrange(grobs = grbs, ncol=3)


# Create a table showing the differences between the various estimations
df_table <- data.frame(Approximation = post_beta_mode,
                       Metropolis = colMeans(metropolisPoints[-(1:1000),-1]))
df_table$Difference <- df_table$Approximation - df_table$Metropolis

df_table <- t(df_table)
names_table <- colnames(X)

library(kableExtra)
kable(df_table, "latex", booktabs = T, align = "c", col.names = names_table,
      linesep = "", escape = F,
      caption = "Comparison between Bayesian normal approximation and Metropolis Hasting
      algorithm for the estimation of the $\\beta$ coefficients.") %>%
  column_spec(c(1), border_right = T) %>%
  row_spec(0, bold = T) %>%
  row_spec(2, hline_after = T) %>%
  kable_styling(latex_options = "hold_position", font_size = 7)
```