Bayesian Learning - all labs

lensc874

2019-05-22

Contents

Lab01

Assignment 1: Bernoulli distribution

Let $y1, y2, y3|\theta \sim Bern(\theta)$, and assume that you have obtained a sample with s = 14 successes in n = 20 trials. Assume a Beta(α_0,β_0) prior for θ and let $\alpha_0 = \beta_0 = 2$

1a. Drawing from posterior

Draw random numbers from the posterior $\theta|(y_1, ..., y_n) \sim Beta(\alpha_0 + s, \beta_0 + f)$ and verify graphically that the posterior mean and standard deviation converges to the true values as the number of random draws grows large.

Given information

- $(y_1, ..., y_n)|\theta \sim Bern(\theta)$
- trials: $n = 20$
- successes: $s = 14$
- prior: $\theta \sim Beta(\alpha_0, \beta_0)$
- $\alpha_0 = \beta_0 = 2$
- posterior: $\theta|(y_1, ..., y_n) \sim Beta(\alpha_0 + s, \beta_0 + f)$

The given information will be implemented in R.

```
# Initializing parameters.
n = 20
s = 14
f = n-s
alpha_0 = 2
beta_0 = 2
```

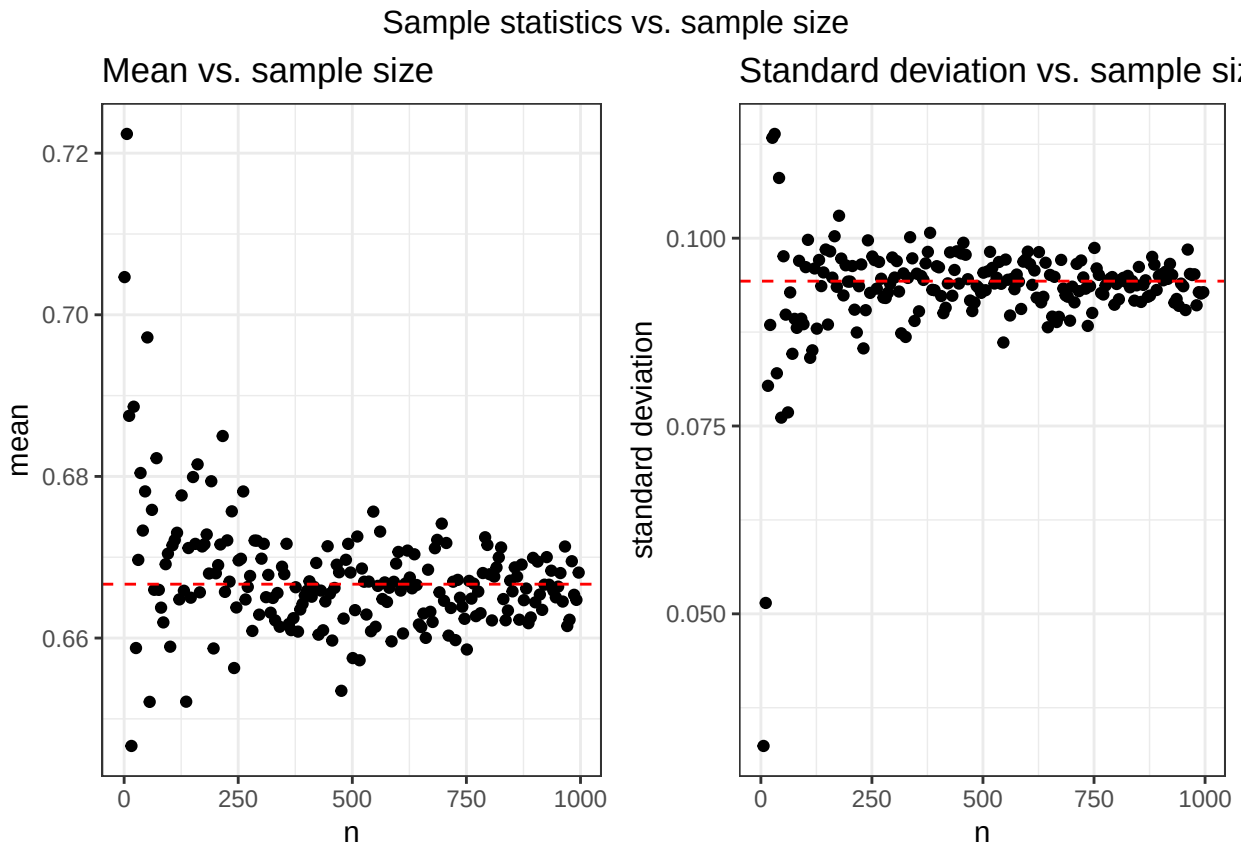
Drawing from posterior and comparing statistics to the ones of true distribution

To draw random values from the posterior, we make use of the `rbeta()`-function. The true mean value and true standard deviation will be calculated by $E(X) = \frac{\alpha}{\alpha+\beta}$ and $sd(X) = \sqrt{\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}}$, respectively.

```
# Calculating true mean and standard deviation.
mean_true = (alpha_0 + s)/((alpha_0 + s)+(beta_0 + f))
sd_true = sqrt(((alpha_0 + s)*(beta_0 + f))/
                 ((((alpha_0 + s)+(beta_0 + f))^2)*((alpha_0 + s)+(beta_0 + f)+1)))
# Generating samples from posterior distribution of different sample sizes and
# Calculating sample mean and sd for samples.
  # Creating emtpy dataframe to store sample statistics for plot.
  sample_stats = data.frame()
  # Generating n samples and storing sample statistics in created data frame.
  for (n in seq(from = 1, to = 1000, by = 5)) {
    theta_samples = rbeta(n = n,
                          shape1 = alpha_0 + s,
                          shape2 = beta_0 + f)
    sample_stats = rbind(sample_stats,
                         cbind(n,
                               mean = mean(theta_samples[1:n]),
                               sd = sd(theta_samples[1:n])))
```

```
  }
# Plotting sample statistics vs. sample size.
library(ggplot2)
library(gridExtra)
plot_mean =
  ggplot(data = sample_stats) +
  geom_point(aes(x = n,
                 y = mean)) +
  geom_hline(yintercept = mean_true, linetype = "dashed", color = "red") +
  theme_bw() +
  labs(title = "Mean vs. sample size",
       x = "n",
       y = "mean")
plot_sd =
  ggplot(data = sample_stats) +
  geom_point(aes(x = n,
                 y = sd)) +
  geom_hline(yintercept = sd_true, linetype = "dashed", color = "red") +
  theme_bw() +
  labs(title = "Standard deviation vs. sample size",
       x = "n",
       y = "standard deviation")
grid.arrange(plot_mean,
             plot_sd,
             ncol = 2,
             top = "Sample statistics vs. sample size")
```

```
Warning: Removed 1 rows containing missing values (geom_point).
```



For both statistics, the mean and standard deviation, it is visible that increasing the sample size leads to to converging to the true values (shown by the dashed lines).

1b. Computing posterior probability for certain θ

Use simulation (nDraws = 10000) to compute the posterior probability $\Pr(\theta < 0.4|\text{y})$ and compare with the exact value [Hint: pbeta()].

In order to simulate 10000 times from the posterior distribution, again the `rbeta()`-function is used. The proportion of sampled values smaller than 0.4 will then be compared with the exact value which will be identified by using the `pbeta()`-function.

```
# Sampling 10000 times from the posterior.
theta_samples = rbeta(n = 10000,
                      shape1 = alpha_0 + s,
                      shape2 = beta_0 + f)
# Calculating probability using samples.
prob_approx = sum(theta_samples < 0.4)/length(theta_samples)
print(paste0("Simulated probability: ", prob_approx))
```

```
[1] "Simulated probability: 0.0035"
```

```
# Calculating true probability using pbeta().
prob_true = pbeta(q = 0.4,
                  shape1 = alpha_0 + s,
                  shape2 = beta_0 + f)
print(paste0("True probability: ", prob_true))
```

```
[1] "True probability: 0.00397268082810898"
```

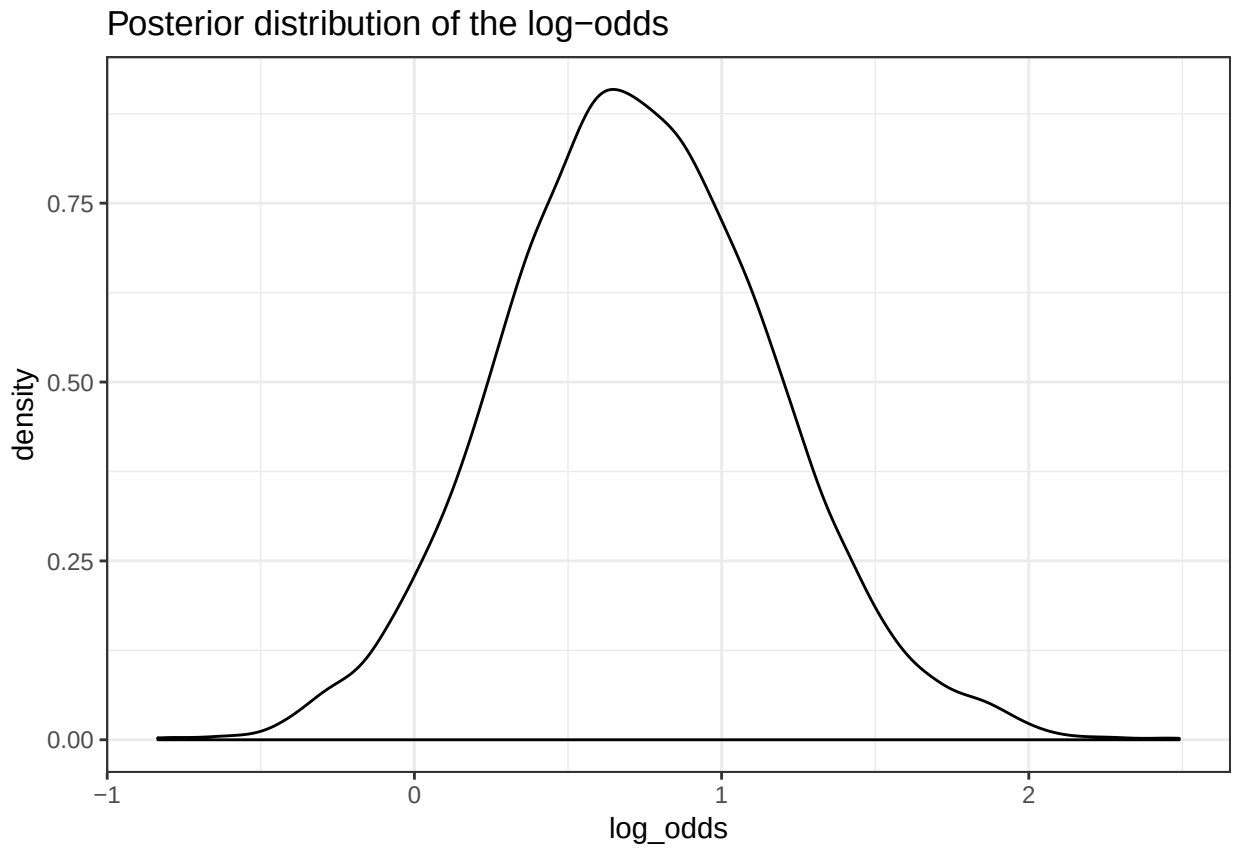
It can be seen that the simulation leads to probability which is very close to the true value.

1c. Computing posterior distribution of the log-odds

Compute the posterior distribution of the log-odds, $\phi = log\frac{\theta}{1-\theta}$ by simulation (nDraws = 10000).

To generate the posterior distribution of the log-odds $\phi = log\frac{\theta}{1-\theta}$, the samples from the posterior distribution for θ obtained in 2b) will be transformed to ϕ-values. As a result, the distribution will be plotted. Also, some statistics are presented.

```
# Transforming sampled theta values to log-odds.
log_odds = log(theta_samples/(1-theta_samples))
# Plotting posterior distribution of log-odds.
ggplot() +
  geom_density(aes(x = log_odds)) +
  labs(title = "Posterior distribution of the log-odds") +
  theme_bw()
```




```
# Printing statistics.
print(density(log_odds))
```

```
Call:
    density.default(x = log_odds)
```

```
Data: log_odds (10000 obs.);    Bandwidth 'bw' = 0.06278

       x                  y
 Min.   :-1.02317   Min.   :0.0000123
 1st Qu.:-0.09781   1st Qu.:0.0052726
 Median : 0.82755   Median :0.0961968
 Mean   : 0.82755   Mean   :0.2699018
 3rd Qu.: 1.75290   3rd Qu.:0.5224779
 Max.   : 2.67826   Max.   :0.9089712
```

Assignment 2: Log-normal distribution and the Gini-coefficient

Assume that you have asked 10 randomly selected persons about their monthly income (in thousands Swedish Krona) and obtained the following ten observations: 14, 25, 45, 25, 30, 33, 19, 50, 34 and 67. A common model for non-negative continuous variables is the log-normal distribution. The log-normal distribution $\log N(\mu, \sigma^2)$ has density function

$$p(y|\mu, \sigma^2) = \frac{1}{y \cdot \sqrt{2\pi\sigma^2}} e[-\frac{(logy - \mu)^2}{2\sigma^2}]$$

for y > 0, $\mu > 0$ and $\sigma^2 > 0$. The log-normal distribution is related to the normal distribution as follows: if $y \sim logN(\mu, \sigma^2)$ then $logy \sim N(\mu, \sigma^2)$. Let $y1, ..., y_n|\mu, \sigma^2 \overset{iid}{\sim} logN(\mu, \sigma^2)$, where $\mu = 3.5$ is assumed to be known but σ^2 is unknown with non-informative prior $p(\sigma^2) \propto 1/\sigma^2$. The posterior for σ^2 is the Inv-χ^2 (n, τ^2) distribution, where

$$\tau^2 = \frac{\sum_{i=1}^{n}(logy_i - \mu)^2}{n}$$

2a. Simulating from posterior of σ^2 and comparing to theoretical posterior distribution

Simulate 10,000 draws from the posterior of σ^2 (assuming $\mu = 3.5$) and compare with the theoretical Inv - $\chi^2(n, \tau^2)$ posterior distribution.

Given information

The following infomation is given:

- $(y_1, ..., y_n)|\mu, \sigma^2 \sim logN(\mu, \sigma^2)$
- $\mu = 3.5$
- σ^2 unknown
- prior: $p(\sigma^2) \propto 1/\sigma^2$ (non-informative prior)
- posterior: $\sigma^2|(y_1, ..., y_n; \mu \sim Inv - \chi^2(n, \tau^2)$ where $\tau^2 = \frac{\sum_{i=1}^{n}(log\ y_i - \mu)^2}{n}$

The given information will be implemented in R.

```
# Initializing parameters.
mu = 3.5
```

Estimating τ^2

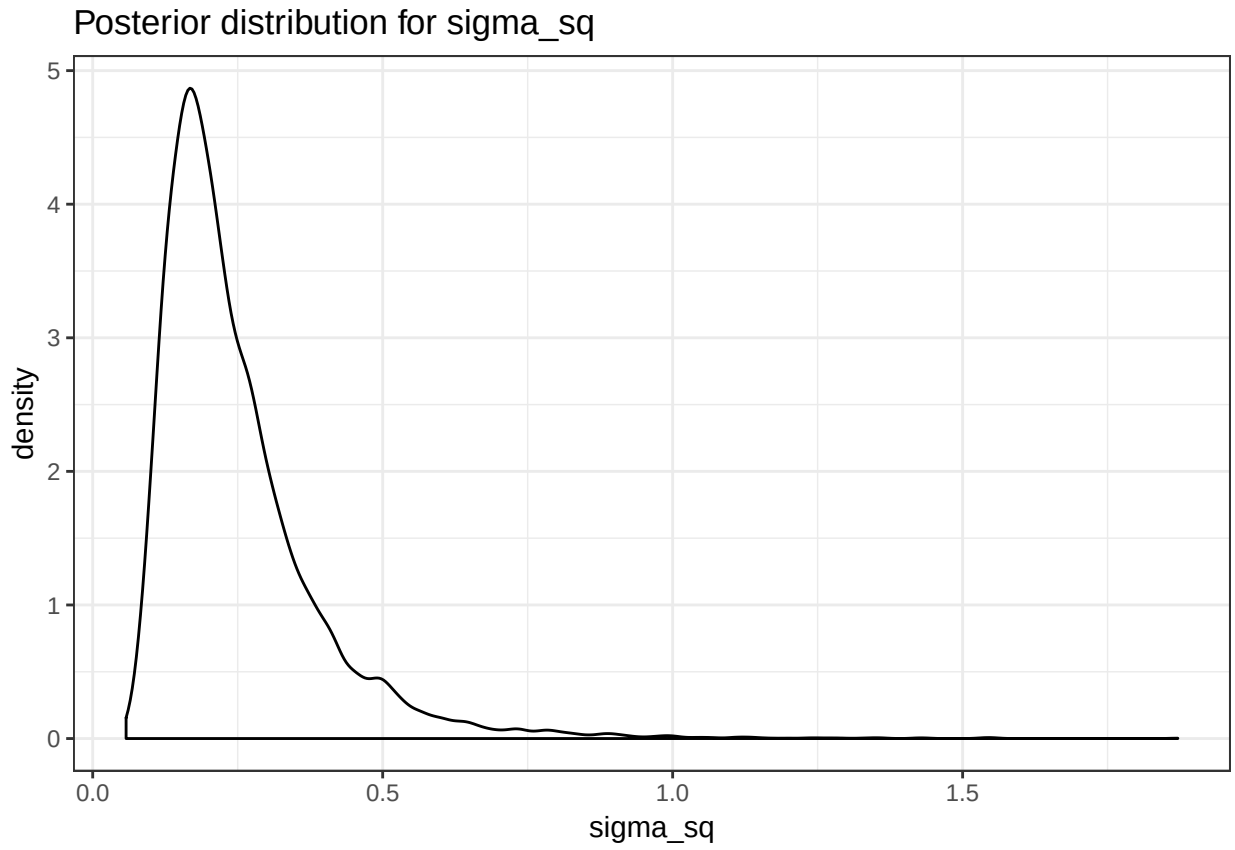
To draw 10000 times from the posterior for $\sigma^2|y_1, ..., y_n$, we first need to estimate the τ^2 by usage of the provided data. Since $\tau^2 = \frac{\sum_{i=1}^{n}(log\ y_i - \mu)^2}{n}$, we can calculate τ^2 as follows:

```
# Calculating tau_sq.
y = c(14, 25, 45, 25, 30, 33, 19, 50, 34, 67)
n = length(y)
tau_sq = sum((log(y)-mu)^2)/n
```

Simulating from posterior and plotting result

In the following, τ^2 can be used to simulate from the posterior $\sigma^2|(y_1, ..., y_n); \mu \sim Inv - \chi^2(n, \tau^2)$. This will be done by first drawing $X \sim \chi^2(n)$. This drawed value will then be used within the formula $\sigma^2 = \frac{n\tau^2}{X}$ which is a draw from $Inv - \chi^2(n, \tau^2)$. This process will be repeated 10000 times. The obtained values for σ^2 will be stored.

```r
sigma_sq = c()
for (i in 1:10000) {
  # Drawing x.
  x = rchisq(n = 1, df = n)
  # Calculating and storing sigma_sq.
  sigma_sq[i] = (n*tau_sq)/x
}
# Plotting simulated posterior distribution.
ggplot() +
  geom_density(aes(x = sigma_sq)) +
  labs(title = "Posterior distribution for sigma_sq") +
  theme_bw()
```



Comparing simulated posterior to theoretical distribution

After the simulated posterior distribution has been plotted, it will be compared to the theoretical distribution. This will be done by a comparison of the mean and the standard deviation. The theoretical mean and standard deviation are obtained with $\frac{n\tau^2}{n-2}$ for $n > 2$ and $\sqrt{\frac{2n^2\tau^4}{(n-2)^2(n-4)}}$ for $n > 4$, respectively.

```r
# Printing statistics of simulated distribution compared to theoretical values.
knitr::kable(
  as.data.frame(
    rbind(
      cbind(Posterior = "Simulation", Mean = mean(sigma_sq), Sd = sd(sigma_sq)),
      cbind(Posterior = "Theory", Mean = n*tau_sq/(n-2), Sd = sqrt((2*n^2*tau_sq^2)/(((n-2)^2)*(n-4))))
    )
```

```
  )
)
```

Posterior	Mean	Sd
Simulation	0.248278471873589	0.138999368821868
Theory	0.247349686559943	0.142807408119353

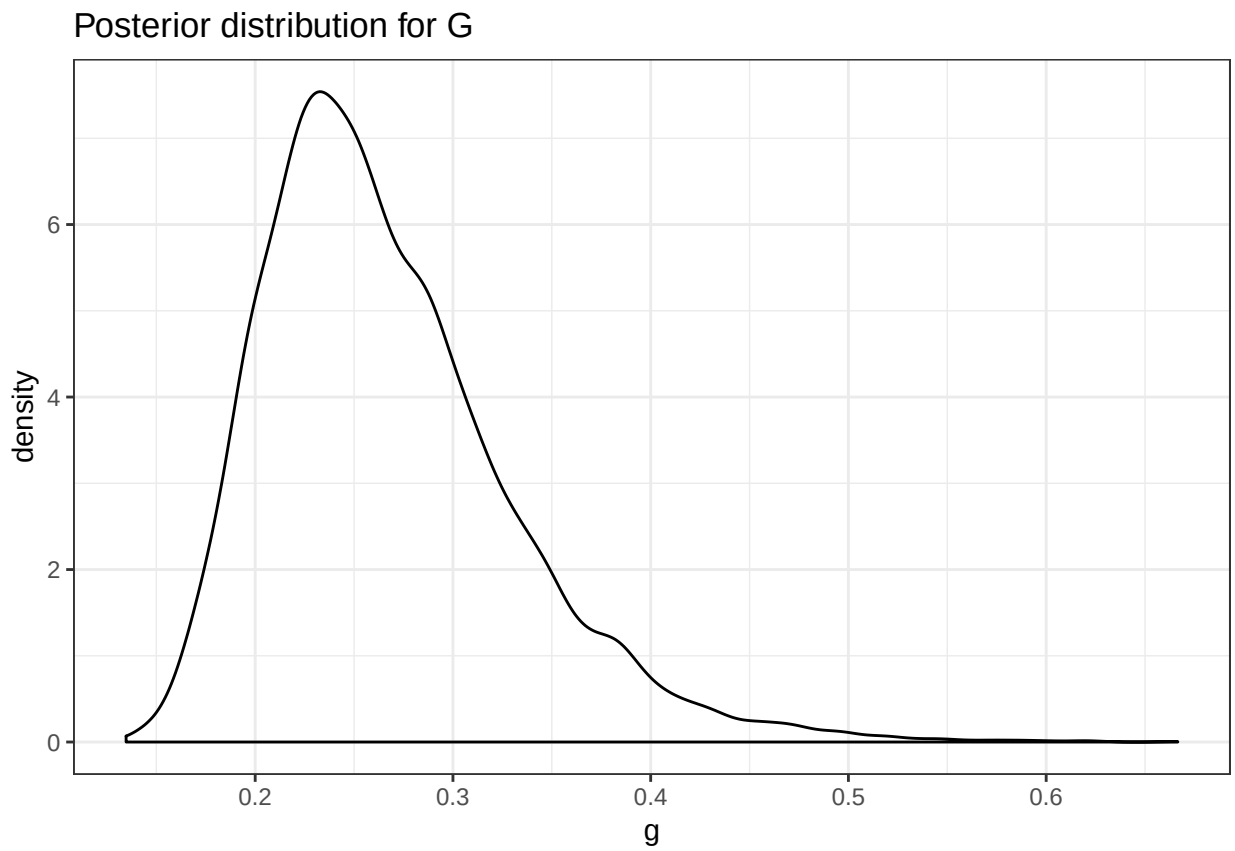
It can be seen that the statistics obtained with the simulation are very close to the theoretical values. Thus, we assume that the simulation of the posterior distribution for σ^2 has been successful.

2b. Computing posterior distribution of Gini coefficient using previous posterior draws

The most common measure of income inequality is the Gini coefficient, G, where $0 \leq G \leq 1$. $G = 0$ means a completely equal income distribution, whereas $G = 1$ means complete income inequality. See Wikipedia for more information. It can be shown that $G = 2\phi(\frac{\sigma}{\sqrt{2}})$ when incomes follow a $\log N(\mu, \sigma^2)$ distribution. $\phi(z)$ is the cumulative distribution function (CDF) for the standard normal distribution with mean zero and unit variance. Use the posterior draws in a) to compute the posterior distribution of the Gini coefficient G for the current data set.

In *2a)*, we simulated the posterior distribution for σ^2 using the current data set. These obtained values for σ^2 now can be used within $G = 2\phi(\sigma/\sqrt{2}) - 1$ to compute the posterior distribution of the Gini coefficient G. Since $\phi(z)$ refers to the CDF of the standard normal distribution, we can make use of the `pnorm(q, mean = 0, sd = 1)`-function where `q` equals to the different values of $\sigma/\sqrt{2}$. The computed distribution will be plotted.

```
# Computing g values.
g = 2 * pnorm(q = sqrt(sigma_sq)/sqrt(2), mean = 0, sd = 1) - 1
# Plotting distribution.
ggplot() +
  geom_density(aes(x = g)) +
  labs(title = "Posterior distribution for G") +
  theme_bw()
```



2c. Computing equal tail and highest posterior density interval

Use the posterior draws from b) to compute a 95 percent equal tail credible interval for G. An 95 percent equal tail interval (a,b) cuts off 2.5 percent of the posterior probability mass to the left of a, and 97.5 percent to the right of b. Also, do a kernel density estimate of the posterior of G using the density function in R with default settings, and use that kernel density estimate to compute a 95 percent Highest Posterior Density interval for G. Compare the two intervals.

Equal tail credible interval

To compute the 95% *equal tail credible interval* for G, the obtained G-values will be checked for duplicated values first.

```
# Checking G values for duplicated values.
any(duplicated(g))
```

```
[1] FALSE
```

Since there are no duplicates, we do not have to group the values and count their frequency, because for every value the frequency equals one. Within the next step, the g-values will be sorted ascendingly and their cumulative percentage will be calculated.

```
# Ordering g.
g_adjusted = g[order(g)]
# Calculating cumulative percentage.
g_adjusted = data.frame(g = g_adjusted,
```

```
                        perc = 1/length(g_adjusted))
g_adjusted$cum_perc = cumsum(g_adjusted$perc)
```

The boundaries of the g-values with a cumulative percentage value equal to 2.5% and equal to 97.5% are identified.

```
# Identifying interval boundaries.
bounds_eq_tail = c(max(g_adjusted[g_adjusted$cum_perc <= 0.025, "g"]),
                   min(g_adjusted[g_adjusted$cum_perc >= 0.975, "g"]))
print(paste0("Lower bound: ", bounds_eq_tail[1], "; Upper bound: ", bounds_eq_tail[2]))
```

```
[1] "Lower bound: 0.174242167225644; Upper bound: 0.417802559449722"
```

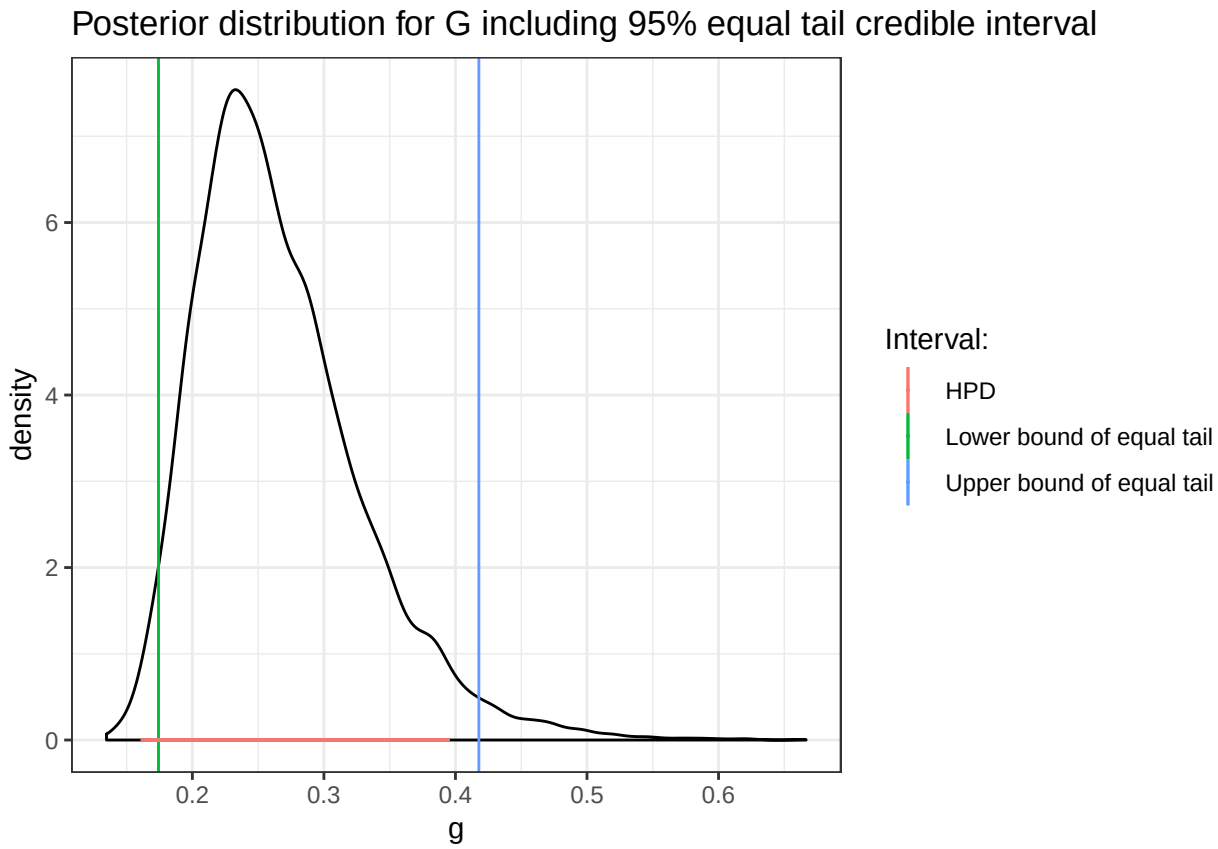
Highest Posterior Density interval

In contrast, the *Highest Posterior Density interval* for G is identified by the 95% highest density values.

```
# Identifying 95% of g-values with highest density.
g_density = data.frame(g = density(g)$x,
                       density = density(g)$y)
# Ordering values by density (highest first).
g_density = g_density[order(g_density$density, decreasing = TRUE),]
# Extracting 95% of g-values with highest density.
g_density$cum_density = cumsum(g_density$density/sum(g_density$density))
g_density = g_density[g_density$cum_density <= 0.95, ]
g_values_hpd = g_density$g
```

Both results will be integrated in the plotted distribution.

```
# Plotting distribution including computed intervals.
ggplot() +
  geom_density(aes(x = g)) +
  geom_vline(aes(xintercept = bounds_eq_tail[1],
                 color = "Lower bound of equal tail")) +
  geom_vline(aes(xintercept = bounds_eq_tail[2],
                 color = "Upper bound of equal tail")) +
  geom_point(aes(x = g_values_hpd, y = 0,
                 color = "HPD"),
             size = 0.02) +
  labs(title = "Posterior distribution for G including 95% equal tail credible interval",
       colour = "Interval:") +
  theme_bw()
```


Posterior distribution for G including 95% equal tail credible interval
density
6
4
2
0
0.2
0.3
0.4
0.5
0.6
g
Interval:
HPD
Lower bound of equal tail
Upper bound of equal tail

Assignment 3: Von Mises distribution

Bayesian inference for the concentration parameter in the von Mises distribution. This exercise is concerned with directional data. The point is to show you that the posterior distribution for somewhat weird models can be obtained by plotting it over a grid of values. The data points are observed wind directions at a given location on ten different days. The data are recorded in degrees and converted into radians: (-2.44, 2.14, 2.54, 1.83, 2.02, 2.33, -2.79, 2.23, 2.07, 2.02)
Assume that these data points are independent observations following the von Mises distribution:

$$p(y|\mu,\kappa) = \frac{e[\kappa \cdot cos(y-\mu)]}{2\pi I_0(\kappa)}, \quad -\pi \leq y \leq \pi$$

where $I_0(\kappa)$ is the modified Bessel function of the first kind of order zero [see ?besselI in R]. The parameter μ $(-\pi \leq y \leq \pi)$ is the mean direction and $\kappa > 0$ is called the concentration parameter. Large κ gives a small variance around μ, and vice versa. Assume that μ is known to be 2:39. Let $\kappa \sim Exponential(\lambda = 1)$ a priori, where λ is the rate parameter of the exponential distribution (so that the mean is $1/\lambda$).

Given information

The following infomation is given:

- ten data values are given (-2.44, 2.14, 2.54, 1.83, 2.02, 2.33, -2.79, 2.23, 2.07, 2.02)
- $p(y_1, ..., y_n)|\mu,\kappa) = \frac{e^{[\kappa * cos(y-\mu)]}}{2\pi I_0(\kappa)}, -\pi \leq y \leq \pi$
- $I_o(\kappa)$ is the modified Bessel function of the first kind of order zero
- $\mu = 2.39$
- κ unknown
- prior: $\kappa \sim Exponential(\lambda = 1)$

The given information will be implemented in R.

```
# Initializing parameters.
mu = 2.39
y = c(-2.44, 2.14, 2.54, 1.83, 2.02, 2.33, -2.79, 2.23, 2.07, 2.02)
```

3a, 3b. Plotting posterior distribution of κ for different values of κ including mode

Plot the posterior distribution of κ for the wind direction data over a fine grid of κ values. Find the (approximate) posterior mode of κ.

Deriving posterior distribution

To plot the posterior distribution of κ, we need to derive the posterior first by the multiplication of the prior and the likelihood. Since we know $p(y_1, ..., y_n)|\mu,\kappa)$ and also the prior distribution, we get the posterior distribution by multiplying the likelihood and the prior. Thus, $p(\kappa|(y_1, ..., y_n), \mu) = \prod_{i=1}^{n} e^{\frac{[\kappa \cos(y_i-\mu)]}{2\pi}} * e^{-\kappa}$. Consequently, we get $p(\kappa|(y_1, ..., y_n), \mu) \propto (\frac{1}{I_0(\kappa)})e^{\kappa \sum_{i=1}^{n} cos(y_i-\mu)-\kappa}$.

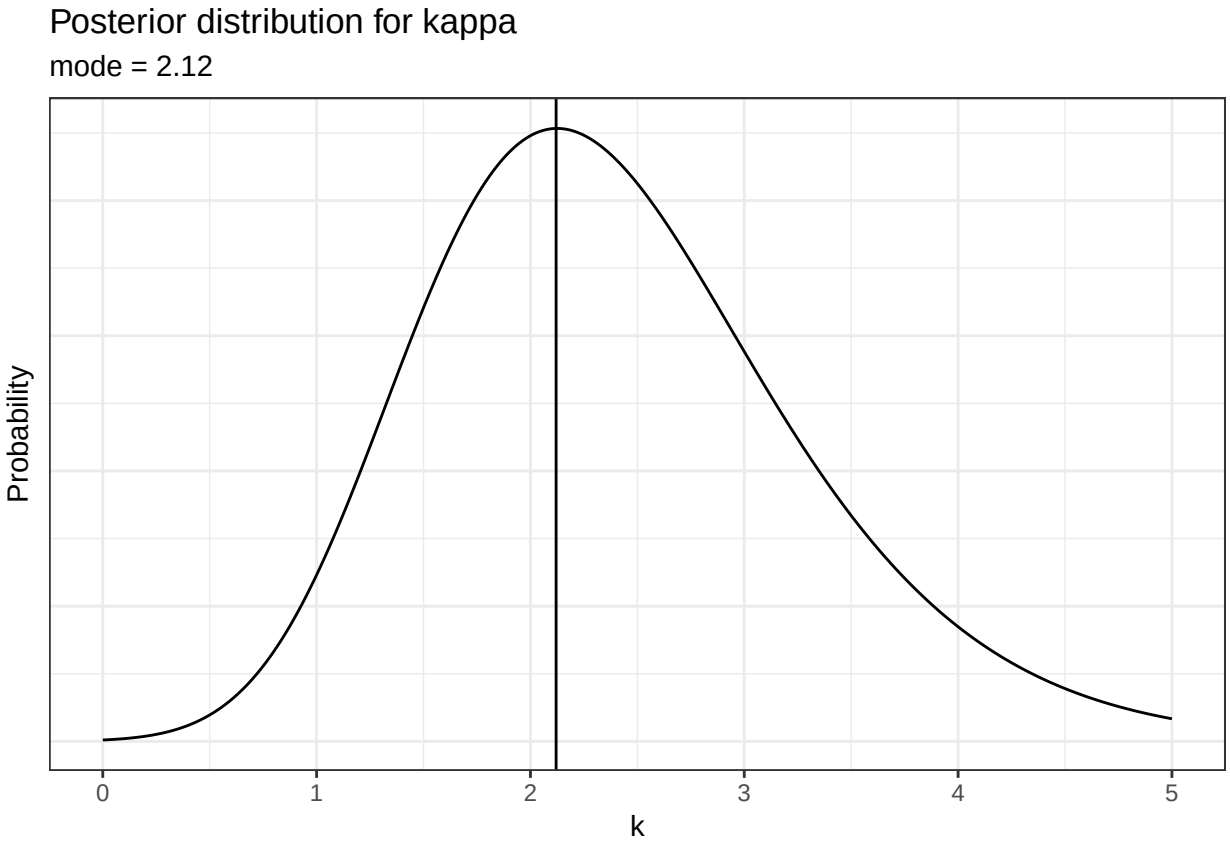
The posterior distribution will be implemented in R. $I_0(\kappa)$ will be implemented with the `besselI(x = kappa, nu = 0)`-function.

```
# Implementing posterior distribtion for kappa.
kappa_posterior = function(y, kappa, mu) {
  n = length(y)
  return((1/besselI(x = kappa, nu = 0))^n*exp(kappa*sum(cos(y-mu))-kappa))
}
```

Computing and plotting posterior for different values of κ including mode

Afterwards, for a sequence of different κ-values, we can compute the posterior distribution using the collected data values. The mode will be selected as the kappa value with the highest probability. The resulting distribution will be plotted.

```
# Computing probabilties for multiple kappa values.
kappa_probs = data.frame()
for (k in seq(from = 0, to = 5, by = 0.01)) {
  kappa_probs = rbind(kappa_probs,
                      cbind(k = k,
                            prob = kappa_posterior(y = y,
                                                   kappa = k,
                                                   mu = mu)))
}
# Plotting distribution.
ggplot() +
  geom_line(data = kappa_probs,
          aes(x = k,
              y = prob)) +
  geom_vline(xintercept = kappa_probs[kappa_probs$prob == max(kappa_probs$prob), "k"]) +
  labs(title = "Posterior distribution for kappa",
       subtitle =
         paste0("mode = ",
                kappa_probs[kappa_probs$prob == max(kappa_probs$prob), "k"]),
       y = "Probability") +
  theme_bw() +
  theme(axis.text.y=element_blank(),
        axis.ticks.y=element_blank())
```


Posterior distribution for kappa
mode = 2.12
Probability
0
1
2
3
4
5
k

Lab02

Assignment 1: Linear and polynomial regression

The dataset TempLinkoping.txt contains daily temperatures (in Celcius degrees) at Malmslott, Linkoping over the course of the year 2016 (366 days since 2016 was a leap year). The response variable is temp and the covariate is

$$\text{time} = \frac{\text{the number of days since beginning of year}}{366}$$

The task is to perform a Bayesian analysis of a quadratic regression

$$\text{temp} = \beta_0 + \beta_1 \cdot \text{time} + \beta_2 \cdot \text{time}^2 + \epsilon, \epsilon \overset{iid}{\sim} N(0, \sigma^2).$$

Reading data

First, the provided data will be read into the R environment.

```
data = read.table("TempLinkoping.txt", header = TRUE, sep = "", dec = ".")
str(data)
```

```
'data.frame':   366 obs. of  2 variables:
 $ time: num  0.00273 0.00546 0.0082 0.01093 0.01366 ...
 $ temp: num  0.1 -4.5 -6.3 -9.6 -9.9 -17.1 -11.6 -6.2 -6.4 -0.5 ...
```

Describing data

The imported dataset contains in total 366 observations. Both variables `time` and `temp` are of class *numeric*. As described, `temp` is given in Celcius degrees. `time` can be calculated by

$$\text{time} = \frac{\text{the number of days since beginning of the year}}{366}$$

and can be therefore interpreted as the time passed up to this moment relative to the entire year. Consequently, the last `time` value equals one.

```
knitr::kable(tail(data), caption = "Extract of the last observations of the provided data.")
```

Table 2: Extract of the last observations of the provided data.

	time	temp
361	0.9863388	5.3
362	0.9890710	2.7
363	0.9918033	-0.4
364	0.9945355	4.3
365	0.9972678	7.0
366	1.0000000	9.3

Implementing function for quadratic regression

The quadratic regression will be implemented in R.

```
# Implementing quadratic regression - function.
quad_regression = function(time, beta_0, beta_1, beta_2, sigma_sq) {
  eps = rnorm(n = 1, mean = 0, sd = sqrt(sigma_sq))
```

```
  temp = beta_0 + beta_1*time + beta_2*(time^2) + eps
  return(temp)
}
```

1a. Determining the prior distribution of the model parameters

Use the conjugate prior for the linear regression model. Your task is to set the prior hyperparameters μ_0, Ω_0, ν_0 and σ_0^2 to sensible values. Start with $\mu_0 = (-10, 100, -100)^T$, $\Omega_0 = 0.01 \cdot I_3$, $\nu_0 = 4$ and $\sigma_0^2 = 1$. Check if this prior agrees with your prior opinions by simulating draws from the joint prior of all parameters and for every draw compute the regression curve. This gives a collection of regression curves, one for each draw from the prior. Do the collection of curves look reasonable? If not, change the prior hyperparameters until the collection of prior regression curves do agree with your prior beliefs about the regression curve. [Hint: the R package mvtnorm will be handy. And use your $Inv - \chi^2$ simulator from Lab 1.]

Implementing joint prior

The conjugate prior for the linear regression is used. It is a joint prior for β and σ^2:

$$\beta|\sigma^2 \sim N(\mu_0, \sigma^2\Omega_0^{-1})$$

$$\sigma^2 \sim Inv - \chi^2(\nu_0, \sigma_0^2)$$

It follows that in total four parameters have to be user-specified (μ_0, Ω_0, ν_0 and σ_0^2).

The conjugate prior will be implemented in R as a function of the user-specified parameters. The function will return a drawn vector for β and the drawn σ^2. To do so, first the value for σ^2 has to be drawn from the *scalded inverse-chi-squared distribution*. We obtain this by usage of the same principle of our simulator from lab01. Precisely, we first draw from $X \sim \chi^2(\nu_0)$ and then compute $\sigma^2 = \frac{\nu_0\sigma_0^2}{X}$. The drawn σ^2 will then be used in the next step as a parameter to draw from a multivariate normal to obtain the β-vector which will be returned. To draw from a multivariate normal, we make use of the `mvtnorm`-package.

```r
# Loading mvtnorm package.
library(mvtnorm)

# Implementing prior.
sim_prior = function(mu_0, omega_0, nu_0, sigma_sq_0) {
  # Drawing sigma_sq from scalded inverse-chi-squared distribution.
    # Drawing x.
    x = rchisq(n = 1, df = nu_0)
    # Calculating sigma_sq using drawn x.
    sigma_sq = (nu_0*sigma_sq_0)/x
  # Drawing beta-vector from multivariate normal using drawn sigma_sq.
  beta_vector = rmvnorm(n = 1, mean = mu_0, sigma = sigma_sq*solve(omega_0))
  # Returning drawn sigma_sq and beta-vector.
  return(list(sigma_sq = sigma_sq, beta_vector = beta_vector))
}
```

Setting prior hyperparameters

Initial values for the parameters are provided and implemented in R.

```r
# Initializing prior parameters.
mu_0_init = as.matrix(c(-10, 100, -100)) # produces 3x1 matrix which is (-10,100,-100)'
omega_0_init = 0.01 * diag(3)
nu_0_init = 4
sigma_sq_0_init = 1
```

Simulating from joint prior

To check if these parameters seem to be reasonable, we simulate draws from the implemented joint prior using the parameters. Within every simulation, we obtain a σ^2 and a β-vector.

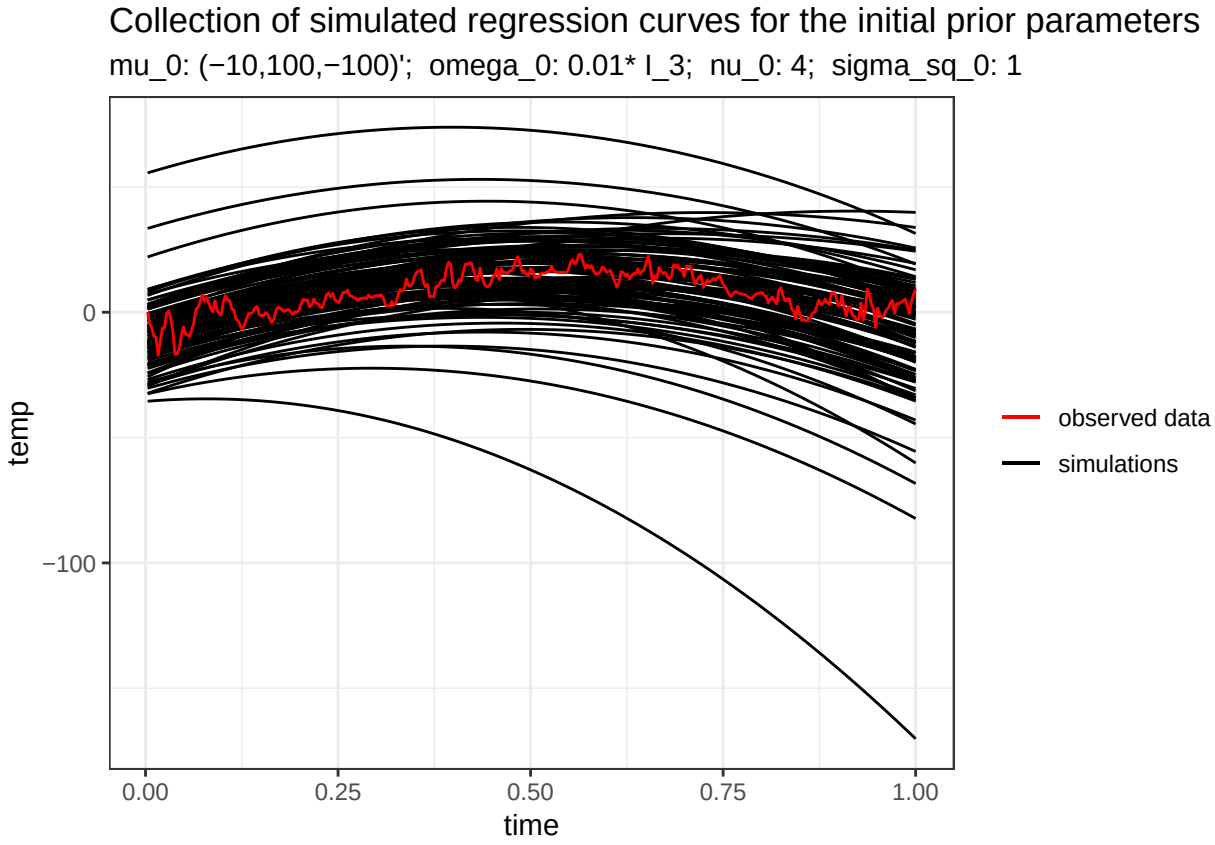
```
# Simulating from prior with initial parameters.
multi_sim_prior = function(n, mu_0, omega_0, nu_0, sigma_sq_0) {
  out = list()
  # Simulating n times.
  for (i in 1:100) {
    out[[i]] = sim_prior(mu_0, omega_0, nu_0, sigma_sq_0)
  }
  return(out)
}
prior_sim_init = multi_sim_prior(100, mu_0_init, omega_0_init, nu_0_init, sigma_sq_0_init)
```

Performing the quadratic regression with simulated draws

With the simulated draws of the β-vector and σ^2, we are able to perform the quadratic regression by usage of the implemented function `quad_regression`. All regression results will be shown graphically in form of regression curves. To assess the quality of the initial prior parameters, we compare the collection of the simulated regression curves with the observed data.

```
# Initializing plot.
library(ggplot2)
initializing_plot = function(mu_0, omega_0, nu_0, sigma_sq_0) {
  out = ggplot() +
  theme_bw() +
  labs(title = "Collection of simulated regression curves for the initial prior parameters",
       subtitle = paste0("mu_0: ", "(", mu_0[1], ",", mu_0[2], ",", mu_0[3], ")'; ",
                         " omega_0: ", omega_0, "* I_3; ",
                         " nu_0: ", nu_0, "; ",
                         " sigma_sq_0: ", sigma_sq_0),
       x = "time",
       y = "temp",
       colour = NULL)
  return(out)
}
prior_plot_init = initializing_plot(mu_0_init, omega_0_init, nu_0_init, sigma_sq_0_init)

# Performing quadratic regression for every simulation and inserting regression curves into plot.
plotting_regr_result = function(prior_sim, prior_plot) {
  for (sim in 1:length(prior_sim)) {
    # Performing quadratic regression.
    temp_hat = quad_regression(time = data$time,
                               beta_0 = prior_sim[[sim]]$beta_vector[1],
                               beta_1 = prior_sim[[sim]]$beta_vector[2],
                               beta_2 = prior_sim[[sim]]$beta_vector[3],
                               sigma_sq = prior_sim[[sim]]$sigma_sq)
    # Inserting regression curve into plot.
    plot_data = data.frame(time = data$time, temp = temp_hat)
    prior_plot = prior_plot +
      geom_line(data = plot_data,
                aes(x = time, y = temp, color = "simulations"))
  }
```

```
  # Inserting observed data into plot.
  plot_data = data.frame(time = data$time, temp = data$temp)
  prior_plot = prior_plot +
    geom_line(data = plot_data,
              aes(x = time, y = temp, color = "observed data")) +
    scale_color_manual(values = c("simulations" = "black",
                                  "observed data" = "red"))
  print(prior_plot)
  return(prior_plot)
}
prior_plot_init = plotting_regr_result(prior_sim_init, prior_plot_init)
```



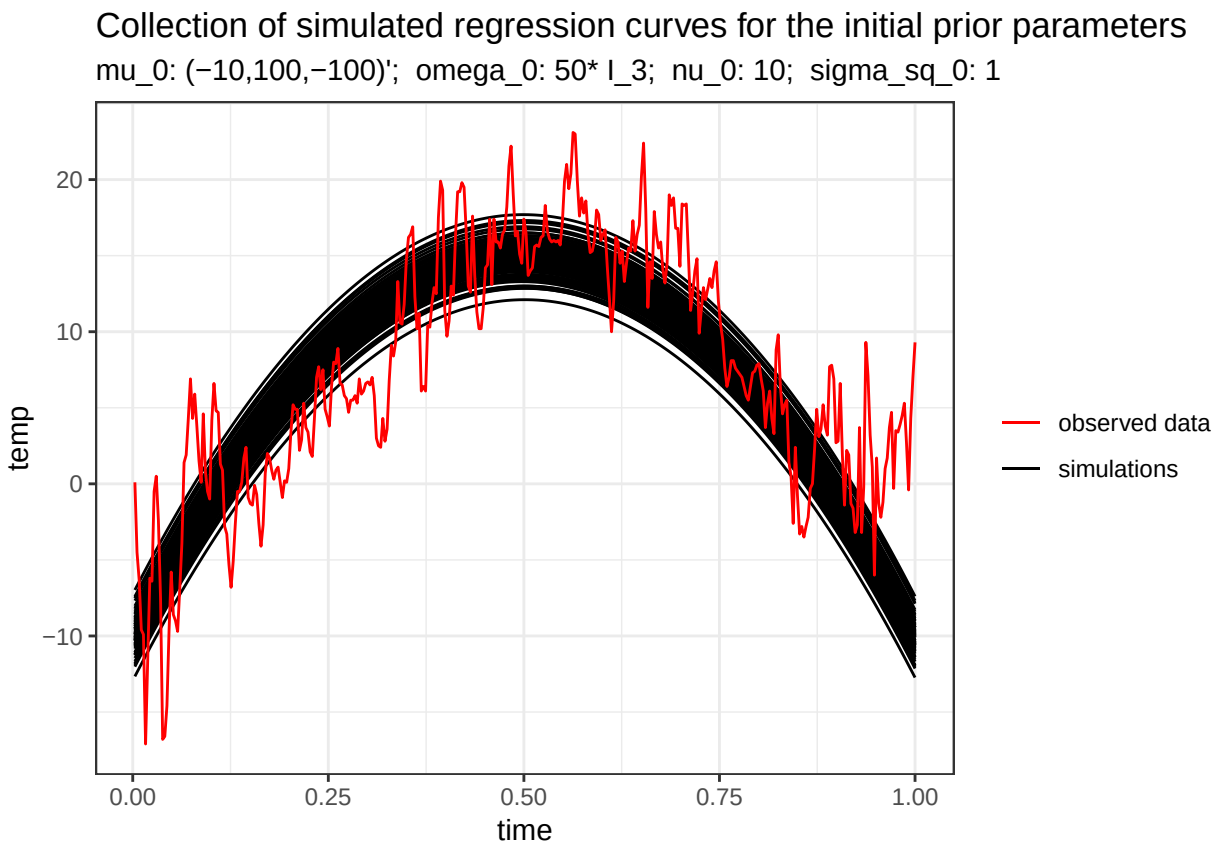
It is obvious that the regression curves simulated using the initial prior parameters are not fitted well to the observed data in multiple cases.

Testing different prior hyperparameters

Consequently, the initial parameters need to be changed to obtain a better regression result. This is done by testing randomly drawn parameters. Within each testing iteration, new parameters are created. With these parameters, the procedure from above is repeated so that within each iteration, a graphical analysis is obtained. We are able to do this by usage of the `readline`-function which stops running the code until we allow R to continue. This gives us the possibility to have a look at the simulated regression curves for the own-specified prior parameters within the iteration to decide if they fit the data well. This procedure allows us to obtain more reasonable parameters graphically.

```
# Initializing user answer for readline-function.
happy = ""

# Repeating process until user specifies that he is happy.
while (happy != "y") {

  # Updating prior parameters.
    # Updating mu_0.
    if (readline("Change mu_0? ") == "y") {
      mu_0_1 = readline("Specify mu_0 for beta_0: ")
      mu_0_2 = readline("Specify mu_0 for beta_1: ")
      mu_0_3 = readline("Specify mu_0 for beta_2: ")
      mu_0_upd = t(as.numeric(c(mu_0_1, mu_0_2, mu_0_3)))
    } else {
      mu_0_upd = mu_0_init
    }
    # Updating omega_0.
    if (readline("Change omega_0? ") == "y") {
      omega_0_upd = readline("Specify omega_0: ")
      omega_0_upd = as.numeric(omega_0_upd)
      omega_0_upd = omega_0_upd * diag(3)
    } else {
      omega_0_upd = omega_0_init
    }
    # Updating nu_0.
    if (readline("Change nu_0? ") == "y") {
      nu_0_upd = readline("Specify nu_0: ")
      nu_0_upd = as.numeric(nu_0_upd)
    } else {
      nu_0_upd = nu_0_init
    }
    # Updating sigma_sq_0.
    if (readline("Change sigma_sq_0? ") == "y") {
      sigma_sq_0_upd = readline("Specify sigma_sq_0: ")
      sigma_sq_0_upd = as.numeric(sigma_sq_0_upd)
    } else {
      sigma_sq_0_upd = sigma_sq_0_init
    }

  # Simulating from prior with updated parameters.
  prior_sim_upd = multi_sim_prior(100, mu_0_upd, omega_0_upd, nu_0_upd, sigma_sq_0_upd)

  # Initializing plot.
  prior_plot_upd = initializing_plot(mu_0_upd, omega_0_upd, nu_0_upd, sigma_sq_0_upd)

  # Performing quadratic regression for every simulation and inserting regression curve into plot.
  prior_plot_upd = plotting_regr_result(prior_sim_upd, prior_plot_upd)

  # Specifying happiness with updated parameters.
  happy = readline("Happy with the regression curves? Then write y.")
}
```

Following this procedure, we were able to obtain the following result:



The subtitle of the plot shows the chosen parameter setting. While changes of μ_0 and σ^2 did not lead to significant improvements and therefore were not changed, especially an increased $\Omega_0 = 50 \cdot I_3$ and a higher $\nu_0 = 10$ results in a much better regression results related to the observed data. Therefore, this parameter setting leads to a collection of regression curves that agree more with our prior beliefs about the regression curve.

Updating prior hyperparameters

To use this prior for the further assigments, the parameters will be implemented in R.

```
# Implementing reasonable prior parameters.
mu_0_opt = mu_0_init
omega_0_opt = 50*diag(3)
nu_0_opt = 10
sigma_sq_0_opt = sigma_sq_0_init
```

1b. Simulating from the joint posterior distribution

Write a program that simulates from the joint posterior distribution of β_0, β_1, β_2 and σ^2. Plot the marginal posteriors for each parameter as a histogram. Also produce another figure with a scatter plot of the temperature data and overlay a curve for the posterior median of the regression function $f(time) = \beta_0 + beta_1 \cdot time + beta_2 \cdot time^2$, computed for every value of time. Also overlay curves for the lower 2.5 percent and upper 97.5 percent posterior credible interval for f(time). That is, compute the 95 percent equal tail posterior probability intervals for every value of time and then connect the lower and upper limits of the interval by curves. Does the interval bands contain most of the data points? Should they?

Given information

The joint posterior for β and σ^2 is as follows:

$$\beta|\sigma^2, y \sim N(\mu_n, \sigma^2\Omega_n^{-1})$$

$$\sigma^2|y \sim Inv - \chi^2(\nu_n, \sigma_n^2)$$

where

$$\mu_n = (X'X + \Omega_0)^{-1}(X'X\hat{\beta} + \Omega_0\mu_0),$$

$$\Omega_n = X'X + \Omega_0,$$

$$\nu_n = \nu_0 + n,$$

$$nu_n\sigma_n^2 = \nu_0\sigma_0^2 + (y'y + \mu_0'\Omega_0\mu_0 - \mu_n'\Omega_n\mu_n)$$

and therefore

$$\sigma_n^2 = \frac{\nu_0\sigma_0^2 + (y'y + \mu_0'\Omega_0\mu_0 - \mu_n'\Omega_n\mu_n)}{\nu_n}.$$

Also we know that

$$\hat{\beta} = (X'X)^{-1}X'y.$$

Implementing parameter values to draw from posterior

Using these formulas and the obtained prior distribution from *1a)*, we can simulate from the posterior distribution. To do so, we make use of a for-loop. Within every iteration, first a value for σ^2 will be drawn. The principle follows the same procedure described and applied as in *1a)* or in lab01. This drawn value will be then used to draw β from the normal distribution. Before implementing the loop, we first implement the necessary parameter values.

```
# Implementing parameter values to draw from sigma_sq and beta.
n = nrow(data)
X = matrix(data = c(rep(1, 366), # intercept.
                    data$time,  # time.
                    (data$time)^2), # time^2.
           ncol = 3)
y = data$temp
beta_hat = solve(t(X)%*%X)%*%t(X)%*%y
mu_n = solve(t(X)%*%X+omega_0_opt)%*%(t(X)%*%X%*%beta_hat+omega_0_opt%*%mu_0_opt)
omega_n = t(X)%*%X+omega_0_opt
nu_n = nu_0_opt + n
sigma_sq_n = as.numeric((nu_0_opt*sigma_sq_0_opt +
                (t(y)%*%y + t(mu_0_opt)%*%omega_0_opt%*%mu_0_opt - t(mu_n)%*%omega_n%*%mu_n))/nu_n)
```

Simulating from joint posterior distribution

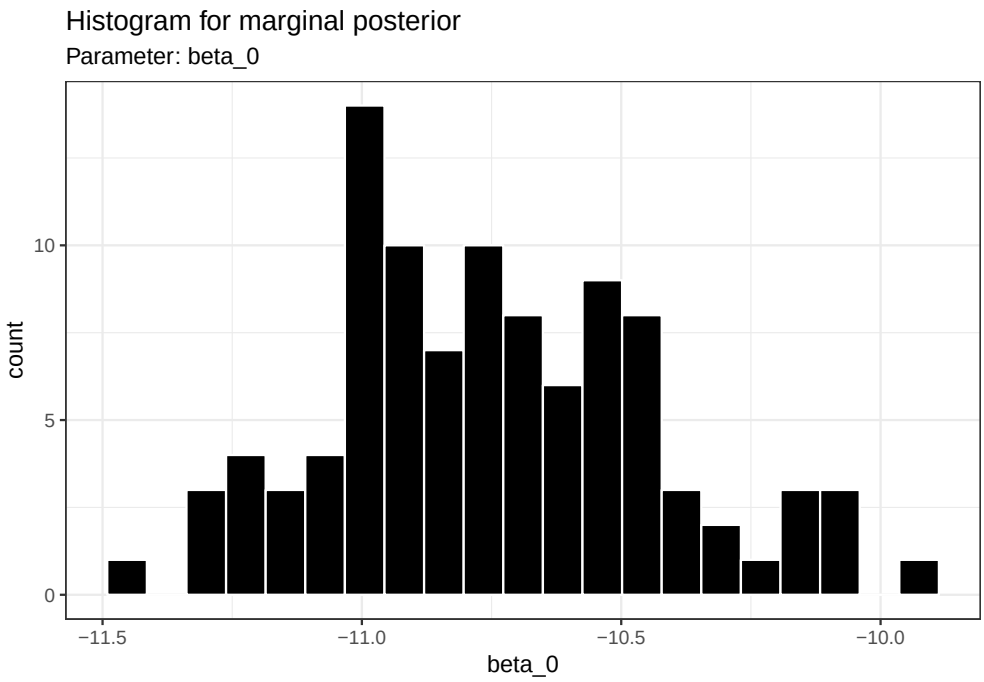
With these parameter values, we are able to draw first from σ^2 and then form $\beta|\sigma^2$. Again, we are simulating 100 times.

```
# Simulating 100 times from posterior.
posterior_sim_all = list()
for (i in 1:100) {
  # Drawing sigma_sq from scalded inverse-chi-squared distribution.
    # Drawing x.
    x = rchisq(n = 1, df = nu_n)
    # Calculating sigma_sq using drawn x.
    sigma_sq = (nu_n*sigma_sq_n)/x
  # Drawing beta-vector from multivariate normal using drawn sigma_sq.
  beta_vector = rmvnorm(n = 1, mean = mu_n, sigma = sigma_sq*solve(omega_n))
  # Returning drawn sigma_sq and beta-vector.
  posterior_sim_all[[i]] = list(sigma_sq = sigma_sq, beta_vector = beta_vector)
}
```

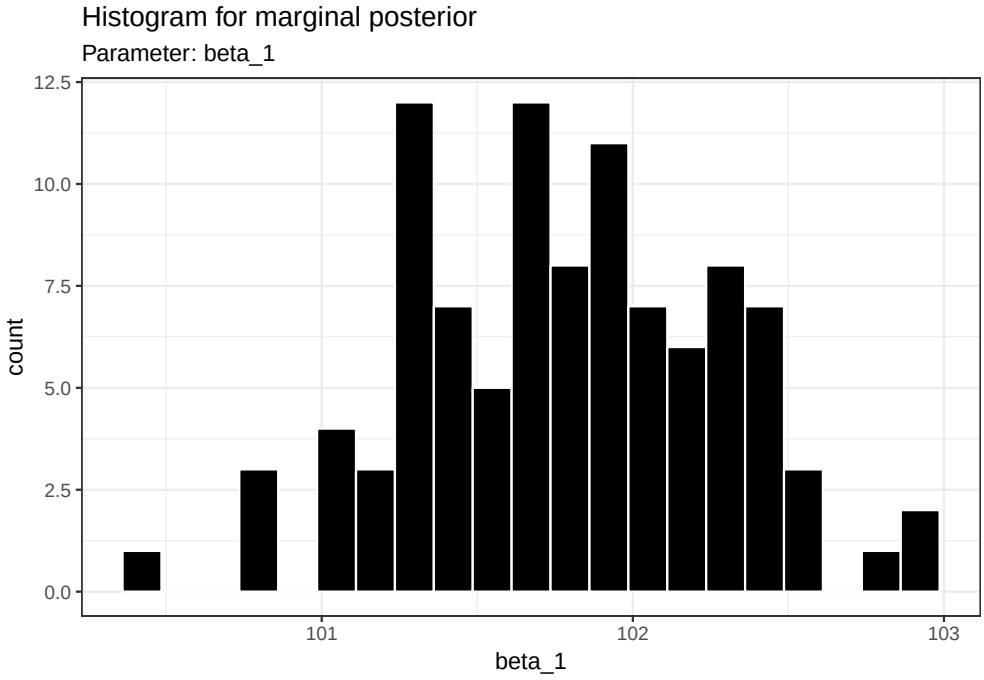
Plotting marginal posteriors for each parameter

Using the simulations, we are able to plot the marginal posteriors for each parameter as a histogram.

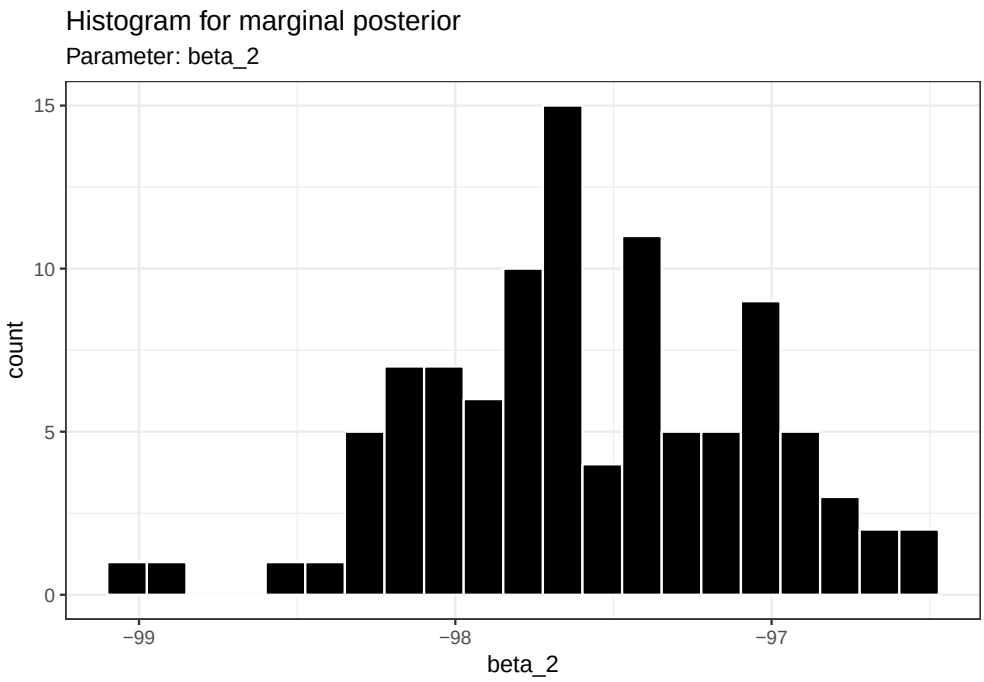
```
# Plotting posteriors for each parameter as histogram.
hist_marg_post = function(param, param_idx) {
  # Preparing data frame for plot.
  # Extracting values of specified parameter from list and stroring values in data frame.
  plot_df = data.frame()
  for (i in 1:length(posterior_sim_all)) {
    plot_df = rbind(plot_df, posterior_sim_all[[i]][[param]][param_idx+1])
  }
  colnames(plot_df) = "var"
  # Extracting plot title from function input.
  if (param == "beta_vector") {
    plot_title = paste0("beta_", param_idx)
  } else {
    plot_title = "sigma_sq"
  }
  # Plotting histogram.
  hist = ggplot() +
    geom_histogram(data = plot_df,
                   aes(x = var),
                   binwidth = (max(plot_df$var)-min(plot_df$var))/(nrow(plot_df)/5),
                   fill = "black",
                   colour = "white") +
    theme_bw() +
    labs(title = "Histogram for marginal posterior",
         subtitle = paste0("Parameter: ", plot_title),
         x = plot_title)
  print(hist)
}
hist_marg_post("beta_vector", 0)
```



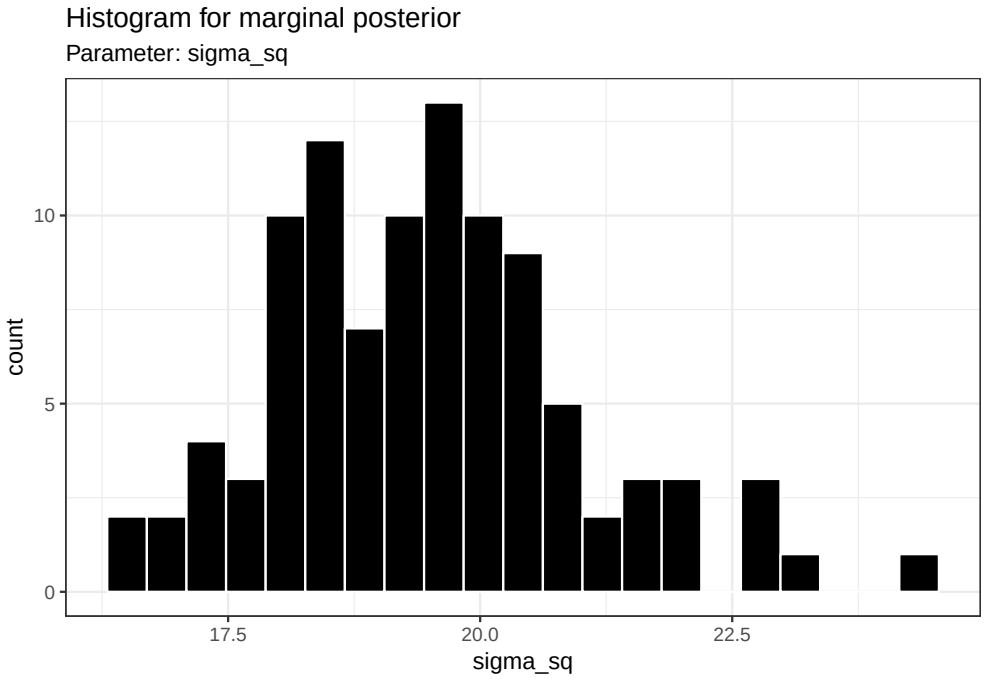

```
hist_marg_post("beta_vector", 1)
```




```
hist_marg_post("beta_vector", 2)
```




```
hist_marg_post("sigma_sq", 0)
```

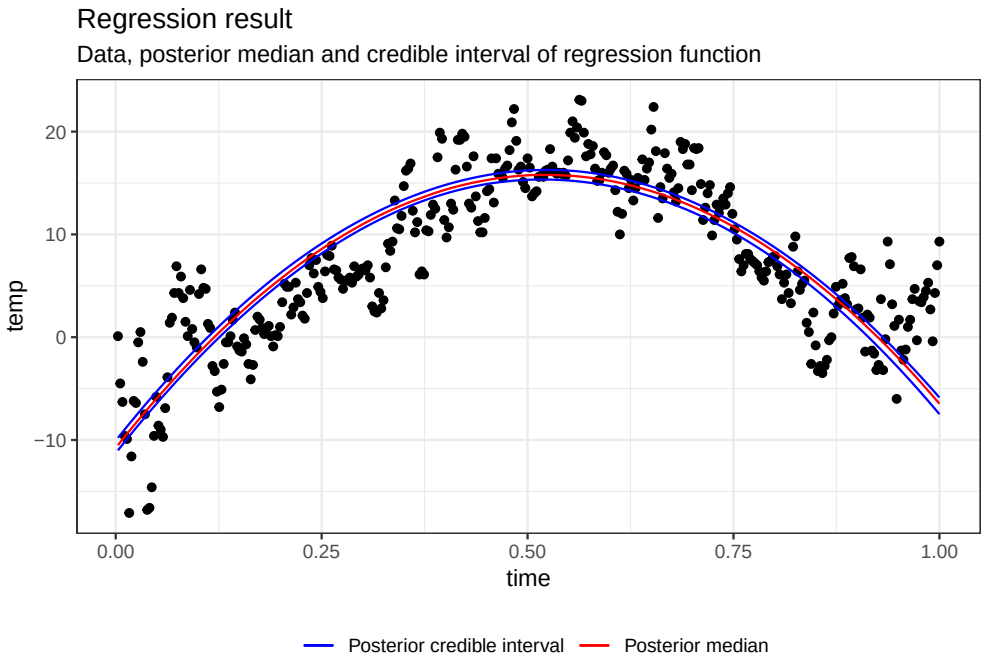


Plotting observed data with posterior results

In the following, a plot with the observed data overlayed by a curve for the posterior median of the regression function $f(time) = \beta_0 + \beta_1 \cdot time + \beta_2 \cdot time^2$ will be created.

```
# Extracting simulated posterior beta values.
posterior_sim_beta = data.frame()
```

```r
for (i in 1:length(posterior_sim_all)) {
  posterior_sim_beta = rbind(posterior_sim_beta,
                             posterior_sim_all[[i]][["beta_vector"]])
}
colnames(posterior_sim_beta) = c("beta_0", "beta_1", "beta_2")
# Performing regression for each value of time using all beta values.
regression_res = data.frame()
for (time in data$time) {
  for (sim in 1:nrow(posterior_sim_beta)) {
    regression_res = rbind(regression_res,
                           cbind(time = time,
                                 temp_hat = posterior_sim_beta[sim, "beta_0"] +
                                   posterior_sim_beta[sim, "beta_1"] * time +
                                   posterior_sim_beta[sim, "beta_2"] * time^2))
  }
}
# Obtaining median, lower limit and upper limit for every value of time.
library(dplyr)
regression_res_agg =
  regression_res %>%
  group_by(time) %>%
  summarise(temp_hat_median = median(temp_hat),
            temp_hat_l_limit = quantile(x = temp_hat, probs = 0.025),
            temp_hat_u_limit = quantile(x = temp_hat, probs = 0.975))
# Plotting observed data with posterior median and credible interval for regression function.
ggplot() +
  # Adding scatter plot of temperature data.
  geom_point(data = data,
             aes(x = time,
                 y = temp)) +
  # Adding curve for posterior median of regression function.
  geom_line(data = regression_res_agg,
            aes(x = time,
                y = temp_hat_median,
                color = "Posterior median")) +
  # Adding curves for posterior credible interval.
  geom_line(data = regression_res_agg,
            aes(x = time,
                y = temp_hat_l_limit,
                color = "Posterior credible interval")) +
  geom_line(data = regression_res_agg,
            aes(x = time,
                y = temp_hat_u_limit,
                color = "Posterior credible interval")) +
  theme_bw() +
  theme(legend.position="bottom") +
  scale_color_manual(values = c("Posterior median" = "red",
                                "Posterior credible interval" = "blue")) +
  labs(title = "Regression result",
       subtitle =  "Data, posterior median and credible interval of regression function",
       colour = NULL)
```



As it can be seen, the interval bands do not contain most of the data points. Since the confidence interval refers to the regression output, it makes sense that it does not inlcude most of the observed data. It rather specifies the range of the output values of the regression.

1c. Simulating from posterior distribution of $\widetilde{x}$

It is of interest to locate the time with the highest expected temperature (that is, the *time* where $f(time)$ is maximal). Let's call this value $\widetilde{x}$. Use the simulations in b) to simulate from the posterior distribution of $\widetilde{x}$. [Hint: the regression curve is a quadratic. You can find a simple formula for $\widetilde{x}$ given β_0, β_1 and β_2.]

It is given that

$$f(time) = \beta_0 + \beta_1 \cdot time + \beta_2 \cdot time^2.$$

To locate the *time* where $f(time)$ is maximal, we calculate

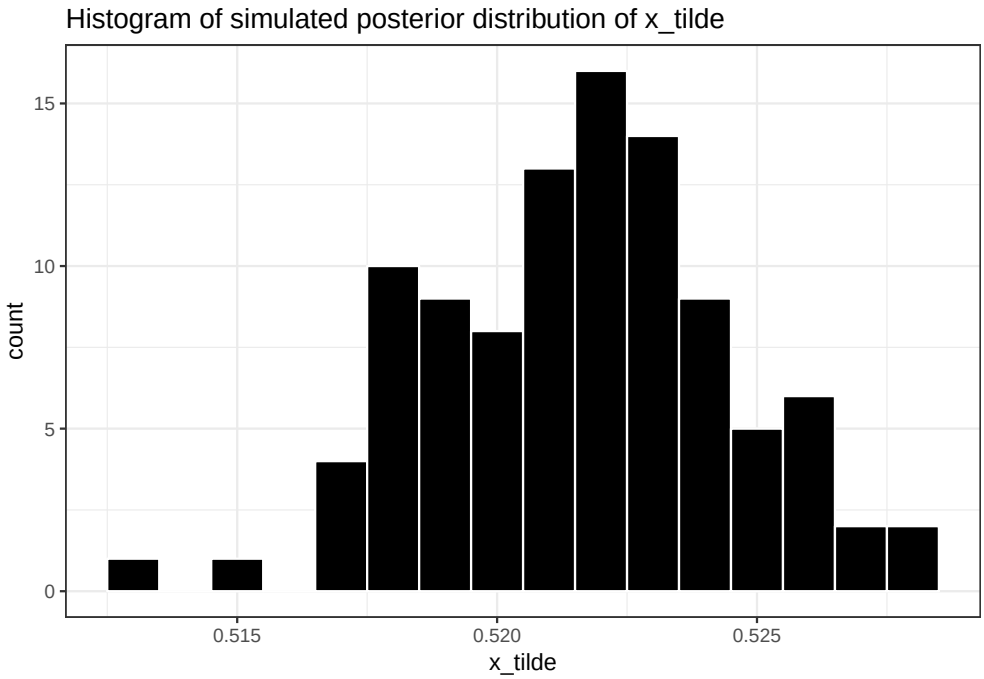
$$\frac{df(time)}{dtime} = \beta_1 + 2\beta_2 time.$$

To obtain the maximum value for *time*, we set the first derivative to zero.

$$\beta_1 + 2\beta_2 time = 0 \Rightarrow time = -\frac{\beta_1}{2\beta_2} = \widetilde{x}$$

With the simulated posterior values for β_1 and β_2 in *2b)*, we are able to simulate from the posterior distribution of $\widetilde{x}$.

```
# Implementing function to compute x_tilde.
sim_x_tilde = function(beta_1, beta_2) {
  return(-beta_1/(2*beta_2))
}
# Simulating from posterior distribution of x_tilde.
posterior_sim_x_tilde = c()
for (sim in 1:nrow(posterior_sim_beta)) {
  posterior_sim_x_tilde[sim] = sim_x_tilde(beta_1 = posterior_sim_beta[sim, "beta_1"],
                                           beta_2 = posterior_sim_beta[sim, "beta_2"])
}
# Plotting histogram of simulated posterior distribution of x_tilde.
ggplot() +
  geom_histogram(aes(x = posterior_sim_x_tilde),
                 fill = "black",
                 colour = "white",
                 binwidth = 0.001) +
  theme_bw() +
  labs(title = "Histogram of simulated posterior distribution of x_tilde",
       x = "x_tilde")
```


Histogram of simulated posterior distribution of x_tilde
count
15
10
5
0
0.515
0.520
0.525
x_tilde

1d. Suggesting prior to estimate a high-order polynomial model

Say now that you want to estimate a polynomial model of order 7, but you suspect that higher order terms may not be needed, and you worry about overfitting. Suggest a suitable prior that mitigates this potential problem. You do not need to compute the posterior, just write down your prior. [Hint: the task is to specify μ_0 and Ω_0 in a smart way.]

To reduce the risk of overfitting while using higher order terms in an estimation of a polynomial model, a possibility is to use a *smoothness/shrinkage/regularization prior*

$$\beta_i|\sigma^2 \overset{iid}{\sim} N(0, \frac{\sigma^2}{\lambda}).$$

It follows that $\mu_0 = 0$ and $\Omega_0 = \lambda I$. Larger λ will then lead to a smoother fit while smaller λ between 0 and 1 will increase the variance of β_i and therefore reduces the risk of overfitting.

Assignment 2: Posterior approximation for classification with logistic regression

Reading data

Within this assignment, we work with the dataset *WomenWork.dat*.

```
# Reading given data.
data = read.table("WomenWork.dat.txt", header = TRUE, sep = "", dec = ".")
```

2a. Implementing logistic regression model

Consider the logistic regression

$$Pr(y = 1|x) = \frac{exp(x^T\beta)}{1 + exp(x^T\beta)}$$

where y is the binary variable with y = 1 if the woman works and y = 0 if she does not. x is a 8-dimensional vector containing the eight features (including a one for the constant term that models the intercept). Fit the logistic regression using maximum likelihood estimation by the command: *glmModel <- glm(Work 0 + ., data = WomenWork, family = binomial).* Note how I added a zero in the model formula so that R doesn't add an extra intercept (we already have an intercept term from the Constant feature). Note also that a dot (.) in the model formula means to add all other variables in the dataset as features. family = binomial tells R that we want to fit a logistic regression.

To Fit the logistic regression using maximum likelihood estimation for the model $Pr(y = 1|x) = \frac{exp(x^T\beta)}{1+exp(x^T\beta)}$, we follow the given instructions.

```
# Fitting logistic regression using maximum likelihood estimation.
glm_model = glm(Work ~ 0 + ., data = data, family = binomial)
```

2b. Approximating the posterior distribution of β with a multivariate normal distribution

Now the fun begins. Our goal is to approximate the posterior distribution of the 8-dim parameter vector β with a multivariate normal distribution

$$\beta|y, X \sim N\left(\widetilde{\beta}, J_y^{-1}(\widetilde{\beta})\right)$$

where $\widetilde{\beta}$ is the posterior mode and $J(\widetilde{\beta}) = -\frac{\partial^2 lnp(\beta|y)}{\partial\beta\partial\beta^T}|_{\beta=\widetilde{\beta}}$ is the observed Hessian evaluated at the posterior mode. Note that $\frac{\partial^2 lnp(\beta|y)}{\partial\beta\partial\beta^T}$ is an 8x8 matrix with second derivatives on the diagonal and cross-derivatives $\frac{\partial^2 lnp(\beta|y)}{\partial\beta_i\partial\beta_j}$ on the offdiagonal. It is actually not hard to compute this derivative by hand, but don't worry, we will let the computer do it numerically for you. Now, both $\widetilde{\beta}$ and $J(\widetilde{\beta})$ are computed by the optim function in R. See my code https://github.com/mattiasvillani/BayesLearnCourse/raw/master/Code/MainOptimizeSpam.zip where I have coded everything up for the spam prediction example (it also does probit regression, but that is not needed here). I want you to implement you own version of this. You can use my code as a template, but I want you to write your own file so that you understand every line of your code. Don't just copy my code. Use the prior $\beta \sim N(0, \tau^2 I)$, with $\tau = 10$. Your report should include your code as well as numerical values for $\widetilde{\beta}$ and $J_y^{-1}(\widetilde{\beta})$ for the WomenWork data. Compute an approximate 95 percent credible interval for the variable NSmallChild. Would you say that this feature is an important determinant of the probability that a women works?

Compting $\widetilde{\beta}$ and observed Hessian

In the next step, the goal is to approximate the posterior distribution of β with a multivariate normal distribution

$$\beta|y, X \sim N\left(\widetilde{\beta}, J_y^{-1}(\widetilde{\beta})\right)$$

where $\widetilde{\beta}$ is the posterior mode and $J(\widetilde{\beta}) = -\frac{\partial^2 lnp(\beta|y)}{\partial\beta\partial\beta^T}|_{\beta=\widetilde{\beta}}$ is the observed Hessian evaluated at the posterior mode. Both $\widetilde{\beta}$ and $J(\widetilde{\beta})$ are computed by the `optim`-function in R. We use the prior $\beta \sim N(0, \tau^2 I)$ with $\tau = 10$.

```
# Defining function to optimize on (log-posterior).
log_posterior = function(beta_vector, y, X, mu, sigma) {
  # Evaluating log of likelihood.
  log_likelihood = sum((X%*%beta_vector)*y -log(1 + exp((X%*%beta_vector))))
  if (abs(log_likelihood) == Inf) log_likelihood = -20000
  # Evaluating log of prior.
  log_prior = dmvnorm(beta_vector, mu, sigma, log=TRUE)
  # Returning log posterior.
  return(log_likelihood + log_prior)
}
# Setting up parameters.
  # Response.
  y = data$Work
  # Features.
  X = as.matrix(data[, -which(colnames(data) == "Work")])
  # Prior.
  tau = 10
  mu_0 = rep(0, ncol(X))
  sigma_0 = tau^2*diag(ncol(X))
  # Initial beta_vector.
  beta_init = rep(0, ncol(X))
```

```r
# Computing beta_mode and hessian.
results_optim = optim(par = beta_init,
                      fn = log_posterior,
                      y = y,
                      X = X,
                      mu = mu_0,
                      sigma = sigma_0,
                      method=c("BFGS"),
                      # Multiplying objective function by -1 to find maximum instead of minimum.
                      control=list(fnscale=-1),
                      hessian=TRUE)
```

The optim-function returns the mode for every β, $\widetilde{\beta}$. Since the function returns $-J_y(\widetilde{\beta})$, we still need to transform this matrix to get the desired $J_y^{-1}(\widetilde{\beta})$ before printing.

```r
# Printing results.
  # Beta_mode.
  knitr::kable(
    data.frame(
      beta_ = seq(from = 0,to = length(results_optim$par)-1),
      posterior_mode = results_optim$par),
    caption = "Numerical values for beta_mode")
```

Table 3: Numerical values for beta_mode

beta_	posterior_mode
0	0.6267288
1	-0.0197911
2	0.1802190
3	0.1675667
4	-0.1445967
5	-0.0820656
6	-1.3591332
7	-0.0246835

```r
  # Adjusted hessian.
  print("Adjusted hessian:")
```

```
[1] "Adjusted hessian:"
```

```r
  -solve(results_optim$hessian)
```

```
             [,1]          [,2]          [,3]          [,4]          [,5]
[1,]  2.266022568  3.338861e-03 -6.545121e-02 -1.179140e-02  0.0457807243
[2,]  0.003338861  2.528045e-04 -5.610225e-04 -3.125413e-05  0.0001414915
[3,] -0.065451206 -5.610225e-04  6.218199e-03 -3.558209e-04  0.0018962893
[4,] -0.011791404 -3.125413e-05 -3.558209e-04  4.351716e-03 -0.0142490853
[5,]  0.045780724  1.414915e-04  1.896289e-03 -1.424909e-02  0.0555786706
[6,] -0.030293450 -3.588562e-05 -3.240448e-06 -1.340888e-04 -0.0003299398
[7,] -0.188748354  5.066847e-04 -6.134564e-03 -1.468951e-03  0.0032082535
[8,] -0.098023929 -1.444223e-04  1.752732e-03  5.437105e-04  0.0005120144
              [,6]          [,7]          [,8]
[1,] -3.029345e-02 -0.1887483542 -0.0980239285
[2,] -3.588562e-05  0.0005066847 -0.0001444223
```

```
[3,] -3.240448e-06 -0.0061345645  0.0017527317
[4,] -1.340888e-04 -0.0014689508  0.0005437105
[5,] -3.299398e-04  0.0032082535  0.0005120144
[6,]  7.184611e-04  0.0051841611  0.0010952903
[7,]  5.184161e-03  0.1512621814  0.0067688739
[8,]  1.095290e-03  0.0067688739  0.0199722657
```

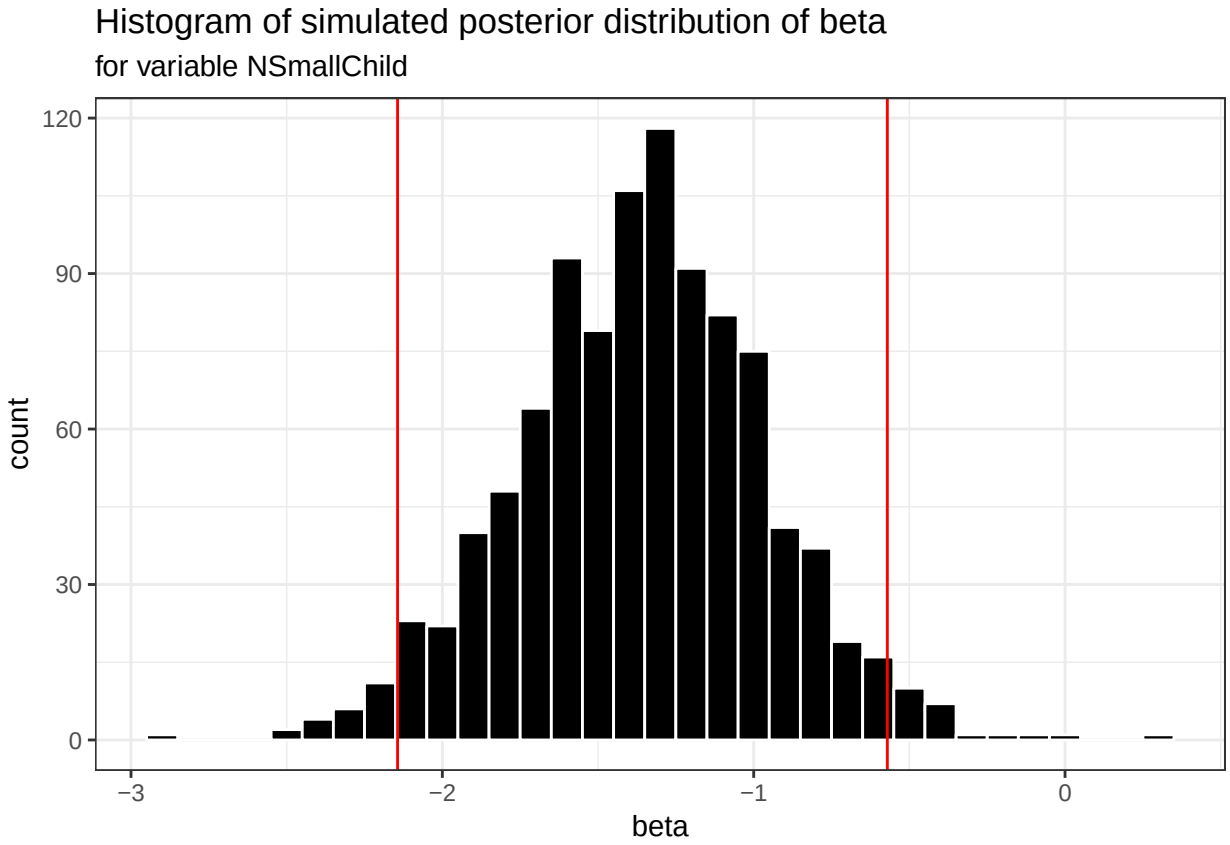
Computing an approximate 95% credible interval for coefficients of variable NSmallChild

To compute an approximate 95% credible interval for the coefficients of the variable *NSmallChild*, we first need to simulate from the posterior. We do this by simulating from the posterior

$$\beta|y, X \sim N\left(\widetilde{\beta}, J_y^{-1}(\widetilde{\beta})\right)$$

by usage of our obtained parameters $\widetilde{\beta}$ and $J_y^{-1}(\widetilde{\beta})$.

```
# Simulating 1000 times from approximated posterior.
posterior_sim_beta_all = rmvnorm(n = 1000,
                                 mean = results_optim$par,
                                 sigma = -solve(results_optim$hessian))
# Extracting simulated posterior beta values for variable NSmallChild.
posterior_sim_beta_nsc = posterior_sim_beta_all[, 7]
# Computing 95% credible interval bounds.
interval_bounds = quantile(x = posterior_sim_beta_nsc,
                           probs = c(0.025, 0.975))
# Plotting simulated beta values with 95% credible interval for variable NSmallChild.
ggplot() +
  geom_histogram(aes(x = posterior_sim_beta_nsc),
                 fill = "black",
                 colour = "white",
                 binwidth = 0.1) +
  geom_vline(xintercept = interval_bounds[1],
             color = "red") +
  geom_vline(xintercept = interval_bounds[2],
             color = "red") +
  theme_bw() +
  labs(title = "Histogram of simulated posterior distribution of beta",
       subtitle = "for variable NSmallChild",
       x = "beta")
```




```
# Printing interval bounds.
knitr::kable(as.data.frame(interval_bounds), caption = "Computed credible interval bounds")
```

Table 4: Computed credible interval bounds

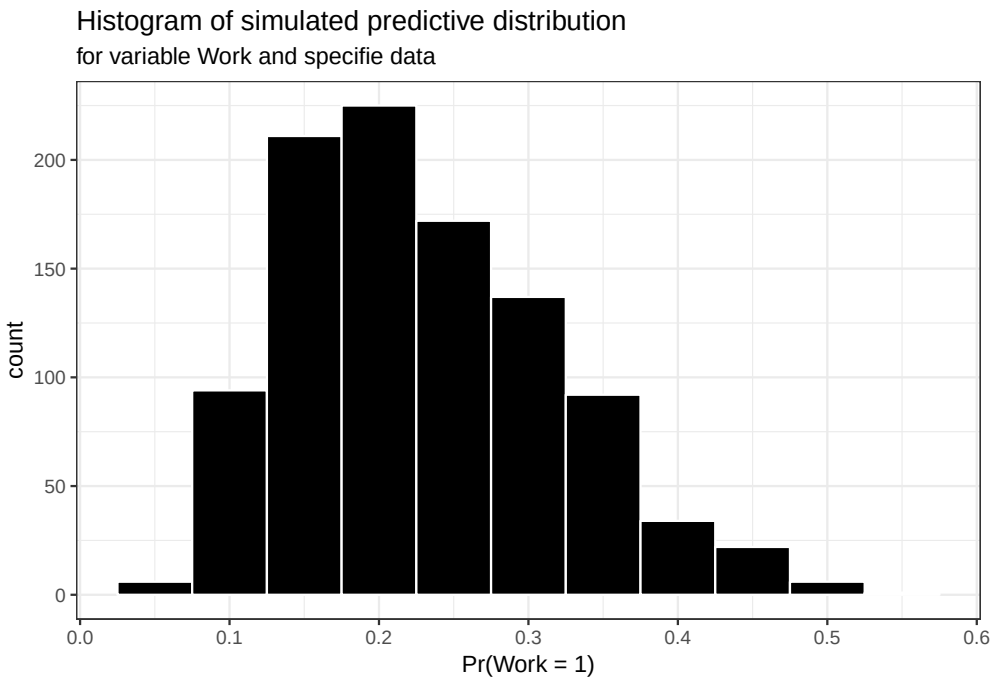
	interval_bounds
2.5%	-2.1438527
97.5%	-0.5707514

2c. Simulating from the predictive distribution of the response variable in a logistic regression

Write a function that simulates from the predictive distribution of the response variable in a logistic regression. Use your normal approximation from 2(b). Use that function to simulate and plot the predictive distribution for the Work variable for a 40 year old woman, with two children (3 and 9 years old), 8 years of education, 10 years of experience. and a husband with an income of 10. [Hint: the R package mvtnorm will again be handy. And remember my discussion on how Bayesian prediction can be done by simulation.]

To perform a simulation from the predictive distribution of the response variable in a logistic regression, we use the drawn posterior β-coefficiencts (`posterior_sim_beta_all`) from *2b)* to predict the *Work*-value. For every drawn β-vector, we calculate $Pr(y = 1|x) = \frac{exp(x^T\beta)}{1+exp(x^T\beta)}$. Since the goal is to predict the *Work*-value for 40-year-old woman with two children (3 and 9 years old), 8 years of education, 10 years of experience and a husband with an income of 10, we use this data for every different drawn β-vector.

```
# Implementing specified data.
x = matrix(c(Constant = 1,
             HusbandInc = 10,
             EducYears = 8,
             ExpYears = 10,
             ExpYears2 = (10/10)^2,
             Age = 40,
             NSmallChild = 1,
             NBigChild = 1),
           ncol = 1)
# Using all drawn posterior beta coeffiecients and specified data vector
# to compute distribution for Pr(y=1|x).
predictive_sim_work = c()
for(sim in 1:nrow(posterior_sim_beta_all)) {
  predictive_sim_work[sim] =
    exp(t(x)%*%posterior_sim_beta_all[sim, ]) /
    (1+exp(t(x)%*%posterior_sim_beta_all[sim, ]))
}
# Plotting histogram of predictive distribution.
ggplot() +
  geom_histogram(aes(x = predictive_sim_work),
                 fill = "black",
                 colour = "white",
                 binwidth = 0.05) +
  theme_bw() +
  labs(title = "Histogram of simulated predictive distribution",
       subtitle = "for variable Work and specifie data",
       x = "Pr(Work = 1)")
```



This is not really the predictive distribution but very much related. It is the predictive probability of y=1 for different posterior draws of beta. The predictive distribution is a distribution over only two values, 0 and 1, so for each probability you should also simulate a binary response variable.

Lab03

Assignment 1: Normal model, mixture of normal model with semi-conjugate prior

The data rainfall.dat consist of daily records, from the beginning of 1948 to the end of 1983, of precipitation (rain or snow in units of $\frac{1}{100}$ inch, and records of zero precipitation are excluded) at Snoqualmie Falls, Washington. Analyze the data using the following two models.

First, the provided data *rainfall.dat* will be read into the R environment.

```
# Storing provided data in R environment as a numeric vector.
data = read.table("rainfall.dat", header = FALSE, sep = "", dec = ".")[,1]
```

1a. Normal model

Assume the daily precipitation $y_1, y_2, y_3...., y_n$ are independent normally distributed, $y_1, y_2, ..., y_n|\mu, \sigma^2 \sim N(\mu, \sigma^2)$ where both μ and σ^2 are unknown. Let $\mu \sim N(\mu_0, \tau_0^2)$ independently of $\sigma^2 \sim Inv - \chi^2(\nu_0, \sigma_0^2)$

i) Implement a Gibbs Sampler code that simulates from the joint posterior $p(\mu, \sigma^2|y_1, y_2, ...y_n)$. The full conditional posteriors are given on the slides from Lecture 7.
ii) Analyze the daily precipitation using your Gibbs sampler in (a)-i. Evaluate the convergence of the Gibbs sampler by suitable graphical methods, for example by plotting the trajectories of the sampled Markov chains.

Given information

For the specified Normal model with the semi-conjugate prior

$$\mu \sim N(\mu_0, \tau_0^2)$$

and

$$\sigma^2 \sim Inv - \chi^2(\nu_0, \sigma_0^2),$$

we know the full-conditional posteriors from lecture 7:

$$\mu|\sigma^2, x \sim N(\mu_n, \tau_n^2)$$

and

$$\sigma^2|\mu, x \sim Inv - \chi^2\left(\nu_n, \frac{\nu_0\tau_0^2 + \sum_{i=1}^n (x_i - \mu)^2}{n + \nu_0}\right).$$

Also, from lecture 2, we know that

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

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

and

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

Setup

To be able to sample from the full conditional posteriors, we have to set up the prior parameters μ_0, τ_0^2, ν_0 and σ_0^2. For μ_0, our best guess is the mean of the provided data. For the other parameters, we do not have any information. Thus, we use initial values of 1 for each of these parameters.

```
# Initializing prior parameters.
mu_0 = mean(data)
tau_0_sq = 1
nu_0 = 1
sigma_0_sq = 1
```

Furthermore, we have to implement n, $\bar{x}$ by usage of the provided data. ν_n is given by $\nu_0 + n$.

```
# Implementing further variables.
n = length(data)
x_bar = mean(data)
nu_n = nu_0 + n
```

In the following, functions which are used to sample from the full conditional posteriors will be implemented following the presented formulas. To draw from the full conditional posterior for σ^2, we need to draw from the scalded inverse-chi-squared distribution. We obtain this by usage of the same principle of our simulators from the previous labs. Precisely, we first draw from $X \sim \chi^2(\nu_n)$ and then compute $\sigma^2 = \frac{\nu_n \left(\frac{\nu_0 \sigma_0^2 + \sum_{i=1}^{n} (x_i - \mu)^2}{n + \nu_0} \right)}{X}$

```
# Implementing functions to enable sampling from full conditional posteriors.
  # mu.
  posterior_mu = function(sigma_sq) {
    # Calculating w.
    w = (n/sigma_sq)/((n/sigma_sq) + (1/tau_0_sq))
    # Calculating mu_n.
    mu_n = w*x_bar+(1-w)*mu_0
    # Calculating tau_n_sq.
    tau_n_sq = 1/((n/sigma_sq)+(1/tau_0_sq))
    # Drawing mu from N(mu_n, tau_n_sq).
    mu = rnorm(n = 1, mean = mu_n, sd = tau_n_sq)
    # Returning draw.
    return(mu)
  }
  # sigma_sq.
  posterior_sigma_sq = function(mu) {
    # Drawing X.
    X = rchisq(n = 1, df = nu_n)
    # Computing sigma_sq.
    sigma_sq = nu_n*((nu_0*sigma_0_sq+sum((data-mu)^2))/(n+nu_0))/X
    # Returning sigma_sq.
    return(sigma_sq)
  }
```

To perform Gibbs sampling, we first need to initialize the $\theta_2, ..., \theta_k$ parameters. Since we only have two parameters in this case, we only need to initialize our θ_2 which is σ^2. To do so, it would be possible to simply select a random value for σ^2. However, we just draw from our prior distribution for σ^2. Again, to sample from the scalded inverse-chi-squared distribution, we follow the same principle as described.

```
# Initializing sigma_sq.
  # Drawing X.
  X = rchisq(n = 1, df = nu_0)
```

```
  # Computing sigma_sq.
  sigma_sq = (nu_0*sigma_0_sq)/X
```

Run

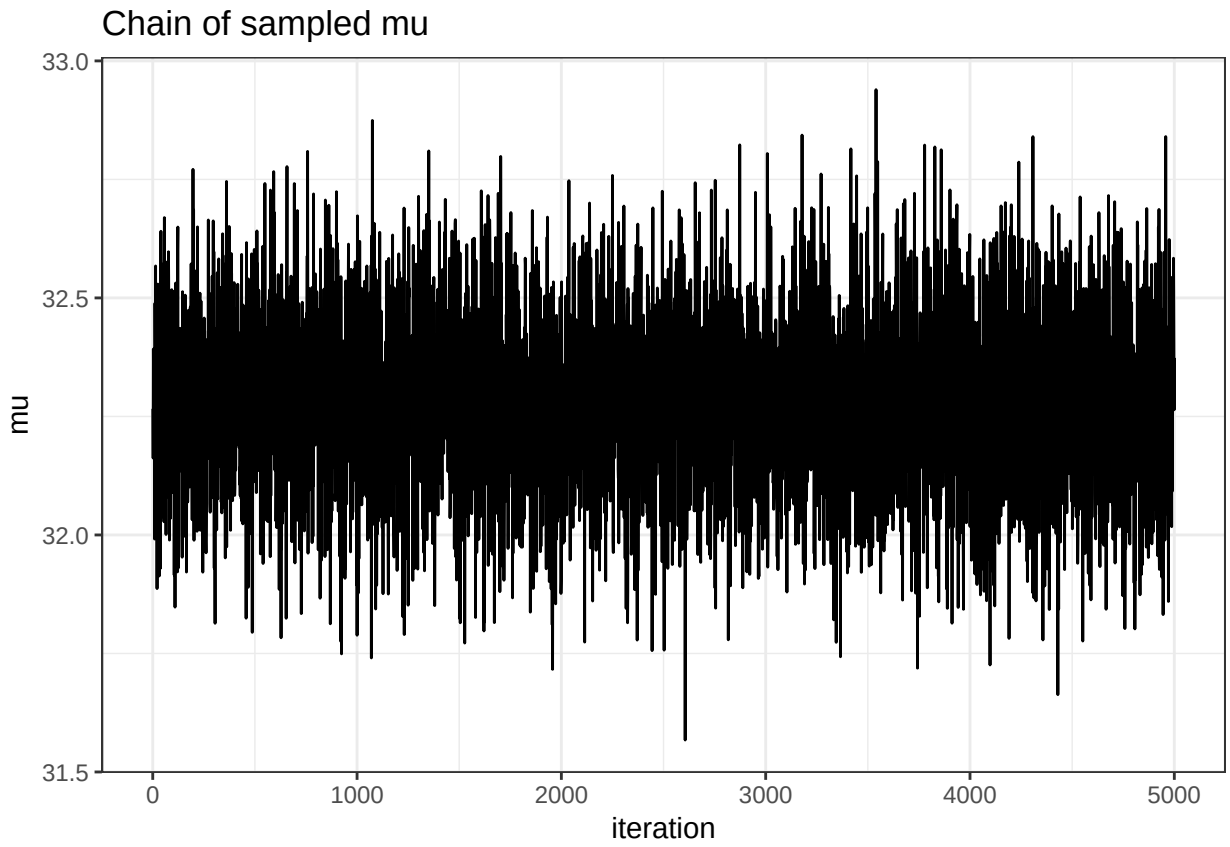
Using both posterior functions and the initialized σ^2, we are now able to perform Gibbs sampling.

```
# Defining number of iterations.
niter = 5000
# Creating objects to store results.
mu_draws = c()
sigma_sq_draws = c()
# Performing gibbs sampling.
for (i in 1:niter) {
  # Sampling and storing mu.
  mu = posterior_mu(sigma_sq)
  mu_draws[i] = mu
  # Sampling and storing sigma_sq.
  sigma_sq = posterior_sigma_sq(mu)
  sigma_sq_draws[i] = sigma_sq
}
```

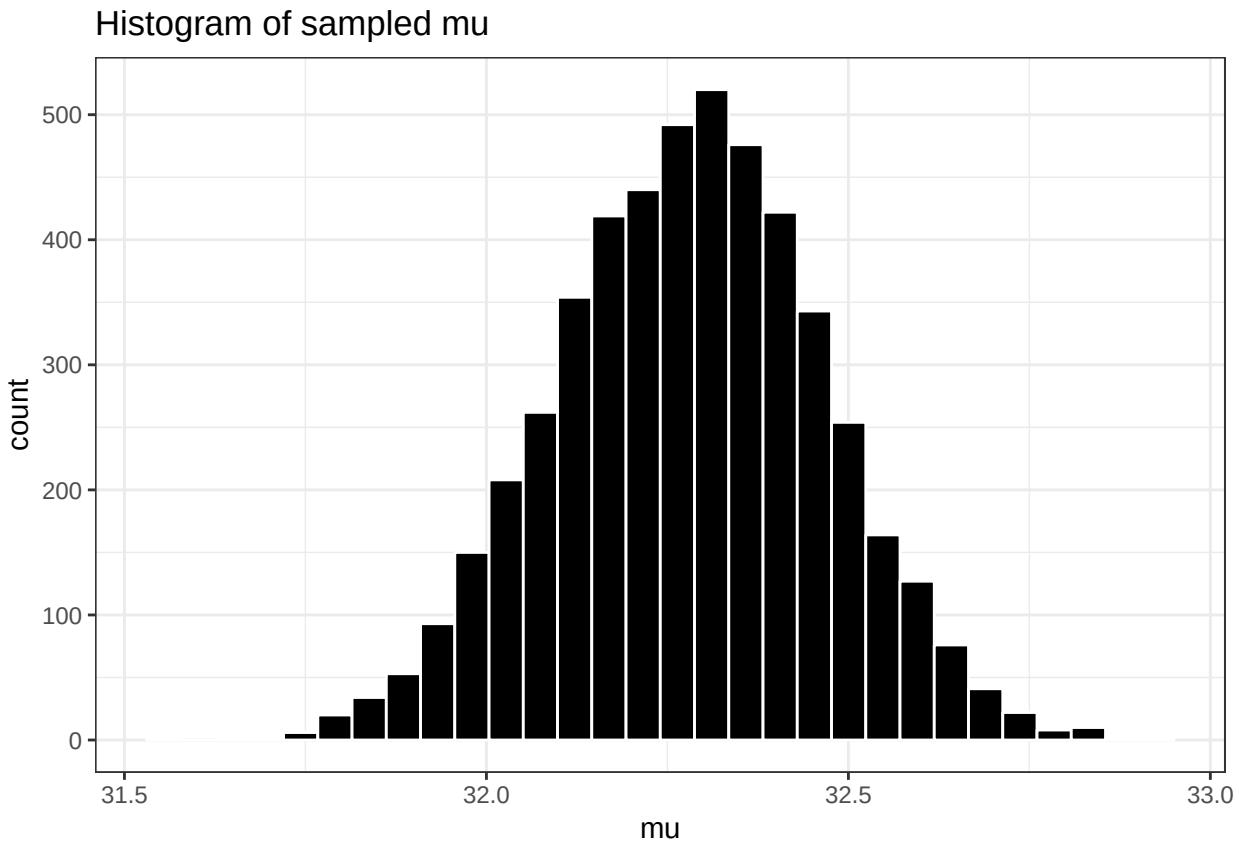
Results

The sampling procedure leads to the following results. For each drawn parameter (μ and σ^2), both the chain and the histogram of the samples are plotted.

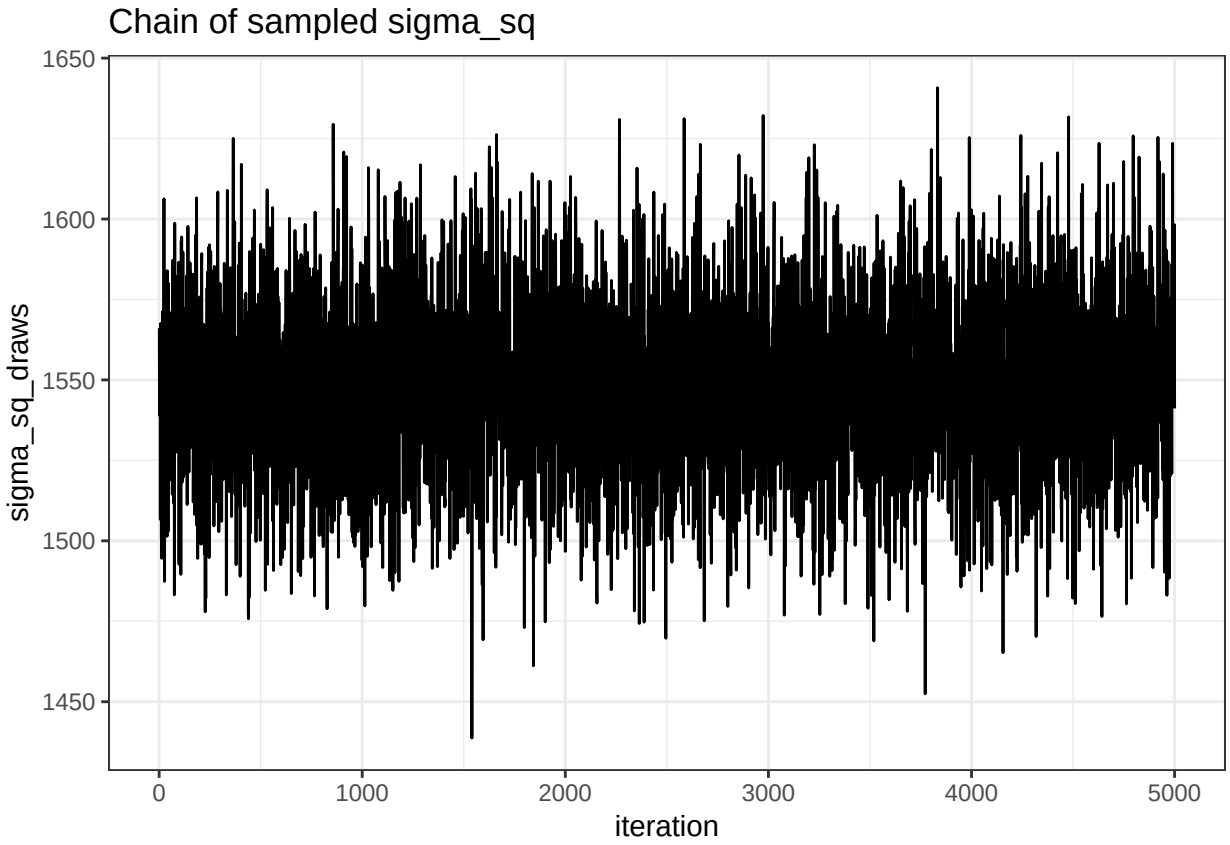
```
# Plotting results of Gibbs sampling.
library(ggplot2)
library(gridExtra)
  # mu.
    # chain.
    ggplot() +
    geom_line(aes(x = seq(from = 1, to = niter), y = mu_draws)) +
    theme_bw() +
    labs(title = "Chain of sampled mu",
         x = "iteration",
         y = "mu")
```



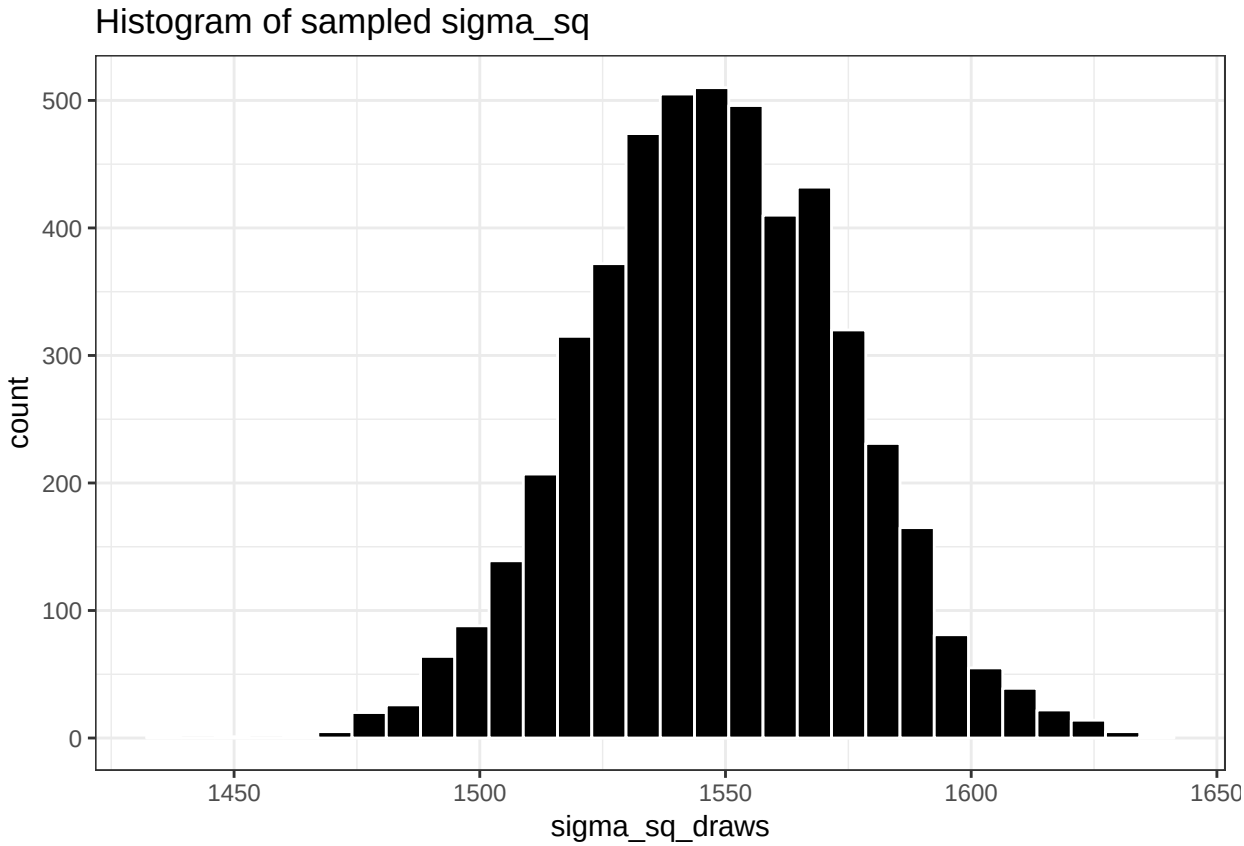

```
# histogram.
ggplot() +
geom_histogram(aes(x = mu_draws),
               fill = "black",
               colour = "white",
               bins = 30) +
theme_bw() +
labs(title = "Histogram of sampled mu",
     x = "mu")
```




```
# sigma_sq.
  # chain.
  ggplot() +
  geom_line(aes(x = seq(from = 1, to = niter), y = sigma_sq_draws)) +
  theme_bw() +
  labs(title = "Chain of sampled sigma_sq",
       x = "iteration",
       y = "sigma_sq_draws")
```




```
# histogram.
ggplot() +
geom_histogram(aes(x = sigma_sq_draws),
               fill = "black",
               colour = "white",
               bins = 30) +
theme_bw() +
labs(title = "Histogram of sampled sigma_sq",
     x = "sigma_sq_draws")
```



The results show that for both parameters μ and σ^2, there is no Burn-in.

1b. Mixture normal model.

Let us now instead assume that the daily precipitation $y_1, y_2, ..., y_n$ follow an iid two-component mixture of normals model:

$$p(y_i|\mu, \sigma^2, \pi) = \pi N(y_i|\mu_1, \sigma_1^2) + (1-\pi)N(y_i|\mu_2, \sigma_2^2)$$

where

$$\mu = (\mu_1, \mu_2) \quad and \quad \sigma^2 = (\sigma_1^2, \sigma_2^2)$$

Use the Gibbs sampling data augmentation algorithm in NormalMixtureGibbs.R (available under Lecture 7 on the course page) to analyze the daily precipitation data. Set the prior hyperparameters suitably. Evaluate the convergence of the sampler.

Given information

The specified mixture normal model is

$$p(y_i|\mu, \sigma^2, \pi) = \pi N(y_i|\mu_1, \sigma_1^2) + (1-\pi)N(y_i|\mu_2, \sigma_2^2),$$

where

$$\mu = (\mu_1, \mu_2) \text{ and } \sigma^2 = (\sigma_1^2, \sigma_2^2).$$

Instead of using the provided code in the file *NormalMixtureGibbs.R*, we create our own programme to perform Gibbs sampling. Within the provided file, another prior is assumed (the prior from lecture 2 where μ_j does not depend on σ_j^2). Therefore, also the full-condtional posteriors differ from the ones provided in lecture 7.

Here, we use the prior

$$\pi \sim Beta(\alpha_1, \alpha_2)$$

and the conjugate prior

$$\mu_j|\sigma_j^2 \sim N\big(\mu_{0j}, \frac{\sigma_j^2}{\kappa_{0j}}\big).$$

$$\sigma_j^2 \sim Inv - \chi^2\big(\nu_{0j}, \sigma_{0j}^2\big)$$

We also define $n_1 = \sum_{i=1}^{n}(I_i = 1)$ and $n_2 = n - n_1$.

According to lecture 7, usage of this prior leads to the following full conditional posteriors:

- $(\pi_1, ..., \pi_k)|I, x \sim Dirichlet(\alpha_1 + n_1, ..., \alpha_k + n_k)$
- $\mu_j|I, \sigma_j^2 \sim N(\mu_{nj}, \frac{\sigma_j^2}{\kappa_{nj}})$
- $\sigma_j^2|I, x \sim Inv - \chi^2(\nu_{nj}, \sigma_{nj}^2)$

where

$$\mu_{nj} = \frac{\kappa_{0j}}{\kappa_{0j} + n_j}\mu_{0j} + \frac{n_j}{\kappa_{0j} + n_j}\bar{x},$$

$$\kappa_{nj} = \kappa_{0j} + n_j,$$

$$\nu_{nj} = \nu_{0j} + n_j$$

and

$$\sigma_{nj}^2 = \frac{\nu_{0j}\sigma_{0j}^2 + (n_j - 1)s^2 + \frac{\kappa_{0j}n_j}{\kappa_{0j}+n_j}(\bar{x} - \mu_{0j})^2}{\nu_n}$$

(see lecture 5).

Furthermore, for our case k = 2 (since we have two components in our model), we can sample I_i as follows:

$$I_i|\pi, \mu_1, \sigma_1^2, \mu_2, \sigma_2^2, x \sim Bern(\theta_i), i = 1, ..., n$$

where

$$\theta_{ij} = \frac{\pi_j \phi(x_i; \mu_j, \sigma_j^2)}{\sum_{r=1}^{k} \pi_r \phi(x_i; \mu_r, \sigma_r^2)}$$

and $\phi(x_i; \mu_j, \sigma_j^2)$ denotes the probability of x_i given $X \sim N(\mu_j, \sigma_j^2)$.

Setup

First, we need to implement our prior beliefs.

```
# Implementing number of components.
n_comp = 2
# Implementing prior parameters.
alpha_j = 10*rep(1, n_comp)
mu_0_j = rep(0, n_comp)
kappa_0_j = rep(3500, n_comp)
nu_0_j = rep(4, n_comp) # degrees of freedom for prior on sigma2
sigma_sq_0_j = rep(var(data), n_comp) # best guess of sigma2
```

Furthermore, functions to use within the sampling process will be implemented.

```
# Implementing function that simulates from Dirichlet distribution
rDirichlet = function(alpha_j) {
  n_alpha = length(alpha_j)
  pi_draws = matrix(NA, n_alpha, 1)
  for (j in 1:n_alpha){
    pi_draws[j] = rgamma(1, alpha_j[j], 1)
  }
  # Dividing every column of pi_draws by the sum of the elements in that column.
  pi_draws = pi_draws/sum(pi_draws)
  return(pi_draws)
}
# Implementing function that simulates from Scaled Inverse Chi Sqaured distribution.
rScaledInvChi2 <- function(n, df, scale){
  return((df*scale)/rchisq(n,df=df))
}
```

To be able to draw from all full conditional posteriors, we need to initialize the paramters I_i (i from observation i), μ_j and σ_j^2.

```
# Initializing parameters for the MCMC.
  # I_i. Random allocation of observations with same probability.
  I = c()
  for (i in 1:length(data)) {
    prob_j = c()
    # Reallocating observation.
    I[i] = which(t(rmultinom(1, size = 1 , prob = rep(1/n_comp, n_comp))) == 1)
  }
  # mu_j.
  mu_j = quantile(data, probs = seq(0,1,length = n_comp))
  # sigma_sq_j.
  sigma_sq_j = rep(var(data), n_comp)
```

Also, at the end, a plot will be created. The plot has to be set up before as well.

```
# 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 = 100)
xGridMin <- min(xGrid)
xGridMax <- max(xGrid)
mixDensMean <- rep(0,length(xGrid))
effIterCount <- 0
ylim <- c(0,2*max(hist(x)$density))
```

```
plotFit <- TRUE
lineColors <- c("blue", "green", "magenta", 'yellow')
sleepTime <- 0.1 # Adding sleep time between iterations for plotting
```

Lastly we still have to setup the number of iterations and we have to initialize the matrices to store the results.

```
# Defining number of iterations.
niter = 200
# Initializing matrices to store results for every iteration.
mu_all = matrix(0, niter, n_comp)
sigma_sq_all = matrix(0, niter, n_comp)
```

Run

With the implemented prior parameters, sampling functions and the initialized parameter values, we are able to perfrom Gibbs sampling.

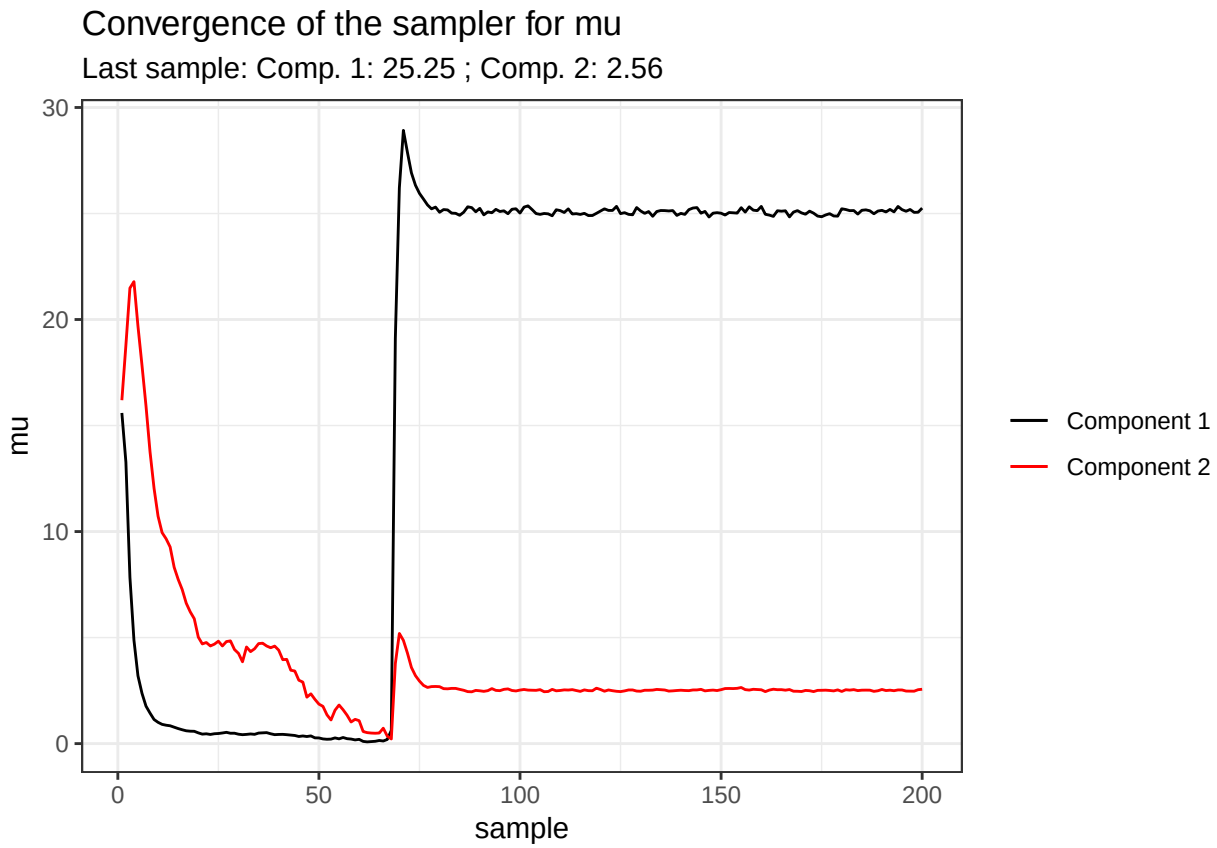
```
# Performing gibbs sampling.
for (i in 1:niter) {

  # Printing iteration number.
  # message(paste('Iteration number:', i))

  # Calculating and printing n_j.
  n_j = c()
  for (j in 1:n_comp) {
    n_j[j] = sum(I == j)
  }
  # print(n_j)

  # Updating parameters.
    # pi.
    pi_j = rDirichlet(alpha_j + n_j)
    # mu.
    for (j in 1:n_comp) {
      # Calculating mu_n_j.
      mu_n_j = (kappa_0_j/(kappa_0_j + n_j))*mu_0_j + n_j/(kappa_0_j + n_j)*mean(data[I == j])
      # Calculating kappa_n_j.
      kappa_n_j = kappa_0_j + n_j
      # Sampling from N(mu_n_j, sigma_sq_j)
      mu_j[j] = rnorm(n = 1, mean = mu_n_j, sd = sigma_sq_j[j] / kappa_n_j)
    }
    mu_all[i, ] = mu_j
    # sigma_sq.
    for (j in 1:n_comp) {
      # Calculating nu_n_j.
      nu_n_j = nu_0_j[j] + n_j[j]
```

```
      # Calculating sigma_sq_n_j.
      sigma_sq_n_j =
        (nu_0_j[j] * sigma_sq_0_j[j] +
        (n_j[j]-1) * sd(data[I == j]) +
        (kappa_0_j[j] * n_j[j]) / (kappa_0_j[j] + n_j[j]) * (mean(data[I == j]) - mu_0_j[j])^2) /
        nu_n_j
      # Sampling from Scaled-Inv-Chi-Sq(nu_n_j, sigma_sq_n_j).
      sigma_sq_j[j] = rScaledInvChi2(n = 1, df = nu_n_j, scale = sigma_sq_n_j)
    }
    sigma_sq_all[i, ] = sigma_sq_j

  # Updating allocation.
  for (obs in 1:length(data)) {
    prob_j = c()
    for (j in 1:n_comp) {
      # Extracting value of current observation.
      x = data[obs]
      # Calculating probability of x belonging to component j.
      prob_j[j] = pi_j[j]*dnorm(x, mean = mu_j[j], sd = sqrt(sigma_sq_j[j]))
    }
    # Reallocating observation.
    I[obs] = which(t(rmultinom(1, size = 1 , prob = prob_j/sum(prob_j))) == 1)
  }

  # Storing data for fitted density against data histogram.
  # Plotting every iteration is uncommented.
  if (plotFit && (i%%1 == 0)){
    effIterCount <- effIterCount + 1
    # hist(data, breaks = 20, freq = FALSE, xlim = c(xGridMin,xGridMax),
    #     main = paste("Iteration number", i), ylim = ylim)
    mixDens <- rep(0,length(xGrid))
    components <- c()
    for (j in 1:n_comp){
      compDens <- dnorm(xGrid,mu_j[j],sd = sqrt(sigma_sq_j[j]))
      mixDens <- mixDens + pi_j[j]*compDens
    #  lines(xGrid, compDens, type = "l", lwd = 2, col = lineColors[j])
      components[j] <- paste("Component ",j)
    }
    mixDensMean <- ((effIterCount-1)*mixDensMean + mixDens)/effIterCount

    #lines(xGrid, mixDens, type = "l", lty = 2, lwd = 3, col = 'red')
    #legend("topleft", box.lty = 1, legend = c("Data histogram",components, 'Mixture'),
    #       col = c("black",lineColors[1:n_comp], 'red'), lwd = 2)
    #Sys.sleep(sleepTime)
  }
}
```

Results

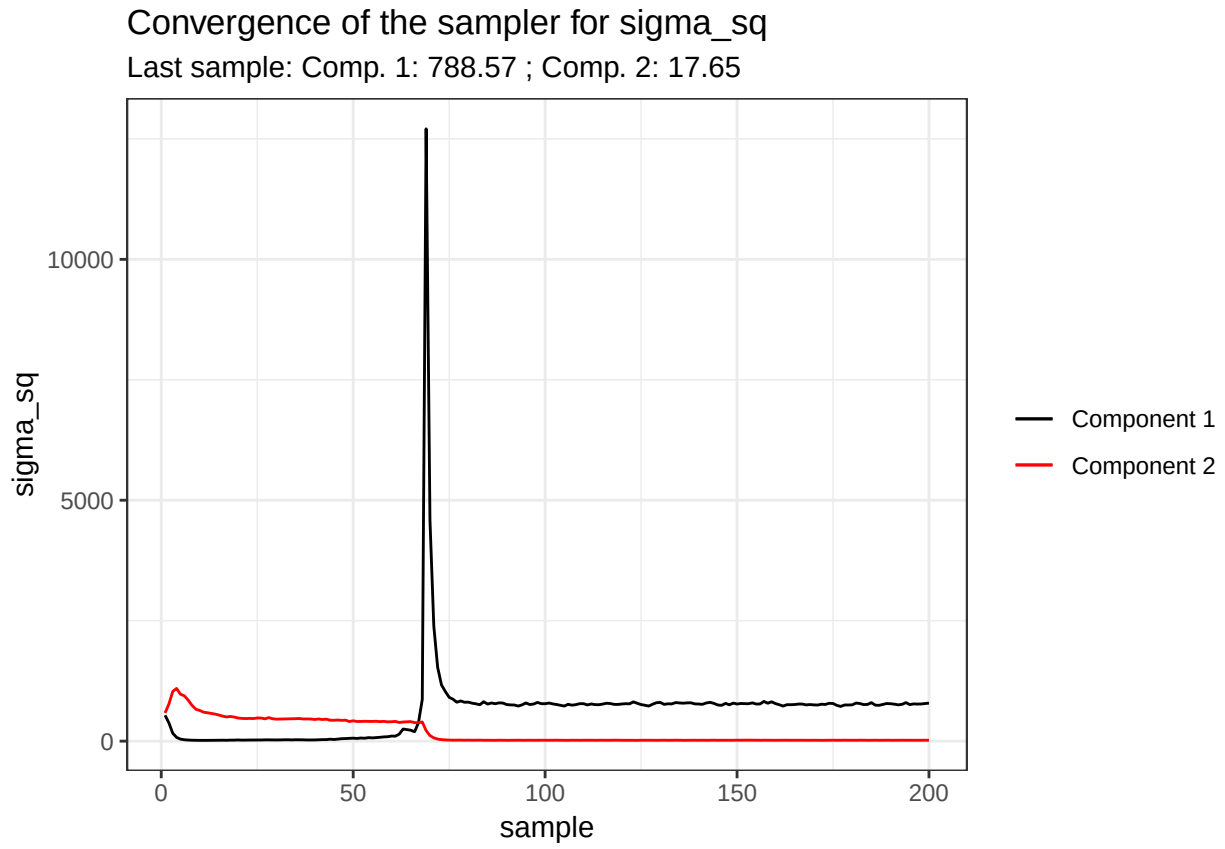
```
# Mu.
ggplot() +
  geom_line(aes(x = seq(1:nrow(mu_all)),
                y = mu_all[,1],
```

```
                color = "Component 1")) +
  geom_line(aes(x = seq(1:nrow(mu_all)),
                y = mu_all[,2],
                color = "Component 2")) +
  scale_color_manual(values = c("Component 1" = "black",
                                "Component 2" = "red")) +
  theme_bw() +
  labs(title = "Convergence of the sampler for mu",
       subtitle = paste0("Last sample: ",
                         "Comp. 1: ",
                         round(mu_all[niter, 1], 2),
                         " ; ",
                         "Comp. 2: ",
                         round(mu_all[niter, 2], 2)),
       x = "sample",
       y = "mu",
       colour = NULL)
```




```
# Sigma_sq.
ggplot() +
  geom_line(aes(x = seq(1:nrow(sigma_sq_all)),
                y = sigma_sq_all[,1],
                color = "Component 1")) +
  geom_line(aes(x = seq(1:nrow(sigma_sq_all)),
                y = sigma_sq_all[,2],
                color = "Component 2")) +
```

```
  scale_color_manual(values = c("Component 1" = "black",
                                "Component 2" = "red")) +
  theme_bw() +
  labs(title = "Convergence of the sampler for sigma_sq",
       subtitle = paste0("Last sample: ",
                         "Comp. 1: ",
                         round(sigma_sq_all[niter, 1], 2),
                         " ; ",
                         "Comp. 2: ",
                         round(sigma_sq_all[niter, 2], 2)),
       x = "sample",
       y = "sigma_sq",
       colour = NULL)
```



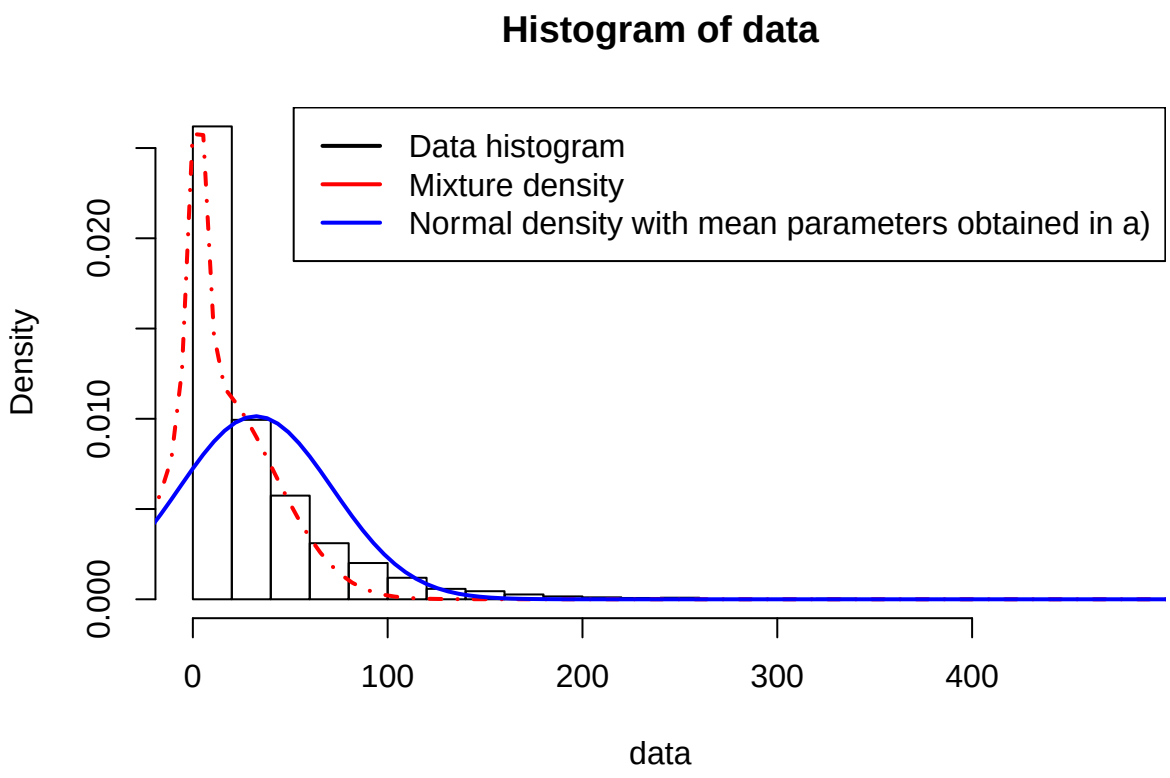
1c. Graphical comparison.

Let $\hat{\mu}$ denote the posterior mean of the parameter μ and correspondingly for the other parameters. Plot the following densities in one figure:

1) a histogram or kernel density estimate of the data.
2) Normal density $N(\hat{\mu}, \hat{\sigma^2})$ in a)
3) Mixture of normals density $p(y_i|\hat{\mu}, \hat{\sigma^2}, \hat{\pi})$ in b)

```
par(mfrow = c(1, 1))
hist(data, freq=FALSE, breaks=20)
```

```
lines(xGrid, mixDensMean, type = "l", lwd = 2, lty = 4, col = "red")
mu = mean(mu_draws)
sigma2 = mean(sigma_sq_draws)
lines(xGrid, dnorm(xGrid, mean = mu, sd=sqrt(sigma2)), type = "l", lwd = 2, col = "blue")
legend("topright", box.lty = 1, legend = c("Data histogram", "Mixture density", "Normal density with mea
```



It becomes obvious that the mixture model fits the data much better.

Assignment 2: Metropolis Random Walk for Poisson regression.

Consider the following Poisson regression model

$$y_i, \beta \sim Poission[\exp(x_i^T, \beta)], i = 1, 2, ..., n$$

where y_i is the count for the ith observation in the sample and x_i is the p-dimensional vector with covariate observations for the ith observation. Use the data set eBayNumberOfBidderData.dat. This dataset contains observations from 1000 eBay auctions of coins. The response variable is nBids and records the number of bids in each auction. The remaining variables are features/covariates (x):
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).

First, the provided data *eBayNumberOfBidderData.dat* will be read into the R environment.

```
# Storing provided data in R environment as a numeric vector.
data = read.table("eBayNumberOfBidderData.dat", header = TRUE, sep = "", dec = ".")
```

2a. Obtaining the maximum likelihood estimator of beta.

Obtain the maximum likelihood estimator of β in the Poisson regression model for the eBay data [Hint: glm.R, don't forget that glm() adds its own intercept so don't input the covariate Const]. Which covariates are significant?

To obtain the maximum likelihood estimator of β in the Poisson regression model, we use the `glm()`-function and specify the family as `poisson`.

```
# Fitting poisson regression using maximum likelihood estimation.
glm_model = glm(nBids ~ ., data = data[, -which(colnames(data) == "Const")], family = poisson)
```

Afterwards, we can identify the significance of the covariates using the `summary()`-function. We assume $\alpha = 0.05$.

```
Covariate significant?
summary(glm_model)$coeff[, 4] < 0.05
```

```
(Intercept) PowerSeller    VerifyID      Sealed     Minblem     MajBlem
       TRUE       FALSE        TRUE        TRUE       FALSE        TRUE
    LargNeg     LogBook MinBidShare
      FALSE        TRUE        TRUE
```

The information about the significant covariates are printed.

```
# Printing beta-coefficients for significant covariates.
knitr::kable(summary(glm_model)$coeff[which(summary(glm_model)$coeff[, 4] < 0.05), ])
```

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	1.0724421	0.0307749	34.847987	0.0000000

	Estimate	Std. Error	z value	Pr(>|z|)
VerifyID	-0.3945165	0.0924258	-4.268468	0.0000197
Sealed	0.4438426	0.0505627	8.778057	0.0000000
MajBlem	-0.2208712	0.0914364	-2.415571	0.0157106
LogBook	-0.1206776	0.0289646	-4.166387	0.0000309
MinBidShare	-1.8940966	0.0712396	-26.587685	0.0000000

2b. Bayesian analysis using approxiamted multivariate normal posterior.

Let's now do a Bayesian analysis of the Poisson regression. Let the prior be $\beta \sim N[0, 100 \cdot (X^T X)^{-1}]$ where X is the nxp covariate matrix. This is a commonly used prior which is called Zellner's g-prior. Assume first that the posterior density is approximately multivariate normal:
In the next step, the goal is to approximate the posterior distribution of β with a multivariate normal distribution

$$\beta|y, X \sim N\left(\widetilde{\beta}, J_y^{-1}(\widetilde{\beta})\right)$$

where $\widetilde{\beta}$ is the posterior mode and $J(\widetilde{\beta}) = -\frac{\partial^2 lnp(\beta|y)}{\partial\beta\partial\beta^T}|_{\beta=\widetilde{\beta}}$ is the observed Hessian evaluated at the posterior mode. Both $\widetilde{\beta}$ and $J(\widetilde{\beta})$ are computed by the 'optim'-function in R. We use the prior $\beta \sim N[0, 100 \cdot (X^T X)^{-1}]$

$$\beta|y \sim N(\widetilde{\beta}, J_y^{-1}(\widetilde{\beta}))$$

where $\widetilde{\beta}$ is the posterior mode and $J_y(\widetilde{\beta})$ is the negative Hessian at the posterior mode. $\widetilde{\beta}$ and $J_y(\widetilde{\beta})$ can be obtained by numerical optimization (optim.R)exactly like you already did for the logistic regression in Lab 2 (but with the log posterior function replaced by the corresponding one for the Poisson model, which you have to code up)

The goal is to implement the approximated posterior multivariate normal. To do so, we need to implement a way to obtain the posterior mode $\widetilde{\beta}$ and the negative Hessian at the posterior mode $J_y(\widetilde{\beta})$ by numerical optimization.

The optim-function can be used to return the mode $\widetilde{\beta}$ for every β. Also, it returns $-J_y(\widetilde{\beta})$ which means that finally we still need to transform this matrix to get the desired $J_y^{-1}(\widetilde{\beta})$ at the end.

```
# Loading mvtnorm package.
library(mvtnorm)

# Defining function to optimize on (log-posterior of poisson).
log_posterior_poi = function(beta_vector, y, X, mu, sigma) {
  # Evaluating log of likelihood for poisson.
  log_likelihood = sum(y * X %*% beta_vector - exp(X %*% beta_vector))
  if (abs(log_likelihood) == Inf) log_likelihood = -20000
  # Evaluating log of prior.
  log_prior = dmvnorm(x = beta_vector,
                      mean = mu,
                      sigma = sigma,
                      log = TRUE)
  # Returning log posterior.
  return(log_likelihood + log_prior)
}
# Setting up parameters.
  # Response.
  y = data$nBids
  # Features.
  X = as.matrix(data[, -which(colnames(data) == "nBids")])
  # Prior.
  mu_0 = rep(0, ncol(X))
  sigma_0 = 100*solve(t(X)%*%X)
  # Initial beta_vector.
  beta_init = rep(0, ncol(X))
# Computing beta_mode and hessian.
results_optim = optim(par = beta_init,
```

```
                      fn = log_posterior_poi,
                      y = y,
                      X = X,
                      mu = mu_0,
                      sigma = sigma_0,
                      method=c("BFGS"),
                      # Multiplying objective function by -1 to find maximum instead of minimum.
                      control=list(fnscale=-1),
                      hessian=TRUE)
# Storing results.
beta_mode = results_optim$par
beta_inv_hessian= -solve(results_optim$hessian)
# Printing results.
  # Beta_mode.
  knitr::kable(data.frame(beta_ = seq(from = 0,to = length(results_optim$par)-1),
                          posterior_mode = results_optim$par),
               caption = "Numerical values for beta_mode")
```

Table 6: Numerical values for beta_mode

beta_	posterior_mode
0	1.0698412
1	-0.0205125
2	-0.3930060
3	0.4435555
4	-0.0524663
5	-0.2212384
6	0.0706968
7	-0.1202177
8	-1.8919850

```
# Adjusted hessian.
print("Adjusted hessian:")
```

```
[1] "Adjusted hessian:"
```

```
beta_inv_hessian
```

```
              [,1]          [,2]          [,3]          [,4]
 [1,]  9.454625e-04 -7.138972e-04 -2.741517e-04 -2.709016e-04
 [2,] -7.138972e-04  1.353076e-03  4.024623e-05 -2.948968e-04
 [3,] -2.741517e-04  4.024623e-05  8.515360e-03 -7.824886e-04
 [4,] -2.709016e-04 -2.948968e-04 -7.824886e-04  2.557778e-03
 [5,] -4.454554e-04  1.142960e-04 -1.013613e-04  3.577158e-04
 [6,] -2.772239e-04 -2.082668e-04  2.282539e-04  4.532308e-04
 [7,] -5.128351e-04  2.801777e-04  3.313568e-04  3.376467e-04
 [8,]  6.436765e-05  1.181852e-04 -3.191869e-04 -1.311025e-04
 [9,]  1.109935e-03 -5.685706e-04 -4.292828e-04 -5.759169e-05
              [,5]          [,6]          [,7]          [,8]
 [1,] -4.454554e-04 -2.772239e-04 -5.128351e-04  6.436765e-05
 [2,]  1.142960e-04 -2.082668e-04  2.801777e-04  1.181852e-04
 [3,] -1.013613e-04  2.282539e-04  3.313568e-04 -3.191869e-04
 [4,]  3.577158e-04  4.532308e-04  3.376467e-04 -1.311025e-04
```

```
[5,]  3.624606e-03  3.492353e-04  5.844006e-05  5.854011e-05
[6,]  3.492353e-04  8.365059e-03  4.048644e-04 -8.975843e-05
[7,]  5.844006e-05  4.048644e-04  3.175060e-03 -2.541751e-04
[8,]  5.854011e-05 -8.975843e-05 -2.541751e-04  8.384703e-04
[9,] -6.437066e-05  2.622264e-04 -1.063169e-04  1.037428e-03
             [,9]
[1,]  1.109935e-03
[2,] -5.685706e-04
[3,] -4.292828e-04
[4,] -5.759169e-05
[5,] -6.437066e-05
[6,]  2.622264e-04
[7,] -1.063169e-04
[8,]  1.037428e-03
[9,]  5.054757e-03
```

2c. Bayesian analysis with using actual posterior using Metropolis algorithm.

Now, let's simulate from the actual posterior of β using the Metropolis algorithm and compare with the approximate results in b). Program a general function that uses the Metropolis algorithm to generate random draws from an arbitrary posterior density. In order to show that it is a general function for any model, I will denote the vector of model parameters by θ. Let the proposal density be the multivariate normal density mentioned in Lecture 8 (random walk Metropolis):

$$\theta_p|\theta^{(i-1)} \sim N(\theta^{(i-1)}, c \cdot \sum)$$

where

$$\sum = J_y^{-1}(\widetilde{\beta})$$

obtained in b). The value c is a tuning parameter and should be an input to your Metropolis function. The user of your Metropolis function should be able to supply her own posterior density function, not necessarily for the Poisson regression, and still be able to use your Metropolis function. This is not so straightforward, unless you have come across function objects in R and the triple dot (...) wildcard argument. I have posted a note (HowToCodeRWM.pdf) on the course web page that describes how to do this in R. Now, use your new Metropolis function to sample from the posterior of β in the Poisson regression for the eBay dataset. Assess MCMC convergence by graphical methods.

First we implement the function `metropolis_sampler()` which will be used to perform the sampling procedure.

Within the implementing procedure, α will calculated as

$$\alpha = min\left(1, \frac{p(\theta_p|y)}{p(^{(i-1)}|y)}\right) = min\left(1, exp\left[log\ p(\theta_p|y) - log\ p(\theta^{(i-1)}|y)\right]\right).$$

Also, instead of using the posterior density values, we make use of the log posterior density, since logs are more stable and avoids problems with too small or large numbers (overflow).

As an input of the function, the user has to specify any posterior density function `logPostFunc` and its parameters, a starting value/vector `theta`, the number of iterations `n` and the tuning parameter `c` related to the variance of the fixed multivariate normal proposal density function.

```
# Implementing metropolis function.

  #metropolis_sampler = function(nBurnIn, n, theta, c, logPostFunc, ...) {
metropolis_sampler = function(logPostFunc, theta, n, c, ...) {

  # Setup.
  theta_prev = theta
  n_accepted = 0
  p = length(theta)
  theta_samples = matrix(NA, n, p)

  # Run.
  #for(i in -nBurnIn : n) {
  for (i in 1:n) {
    # Sample from proposal distribution
    theta_cand = as.vector(rmvnorm(n = 1,
                                   mean = theta_prev,
                                   sigma = c * beta_inv_hessian))
    # Calculate log posteriors
    log_post_cand = logPostFunc(theta_cand, ...)
    log_post_prev = logPostFunc(theta_prev, ...)
```

```
    # Calculate alpha
    alpha = min(1, exp(log_post_cand - log_post_prev))

    # Select sample with probability alpha
    u = runif(1)
    if (u <= alpha){
      theta_samples[i,] = theta_cand
      theta_prev = theta_cand
      n_accepted = n_accepted + 1
    } else {
      theta_samples[i,] = theta_prev
    }
    # Save sample if not burnin
    #if (i>0) theta.samples[i,] = theta.c
  }
  cat("Sampled", n, "samples with an acceptance rate of", n_accepted/n)
  return(theta_samples)
}

# Sampling from posterior using metropolis function.
res = metropolis_sampler(logPostFunc = log_posterior_poi,
                         theta = beta_mode,
                         n = 1000,
                         c = 0.6,
                         mu = mu_0,
                         sigma = sigma_0,
                         X = X,
                         y = y)
```

```
Sampled 1000 samples with an acceptance rate of 0.285
```

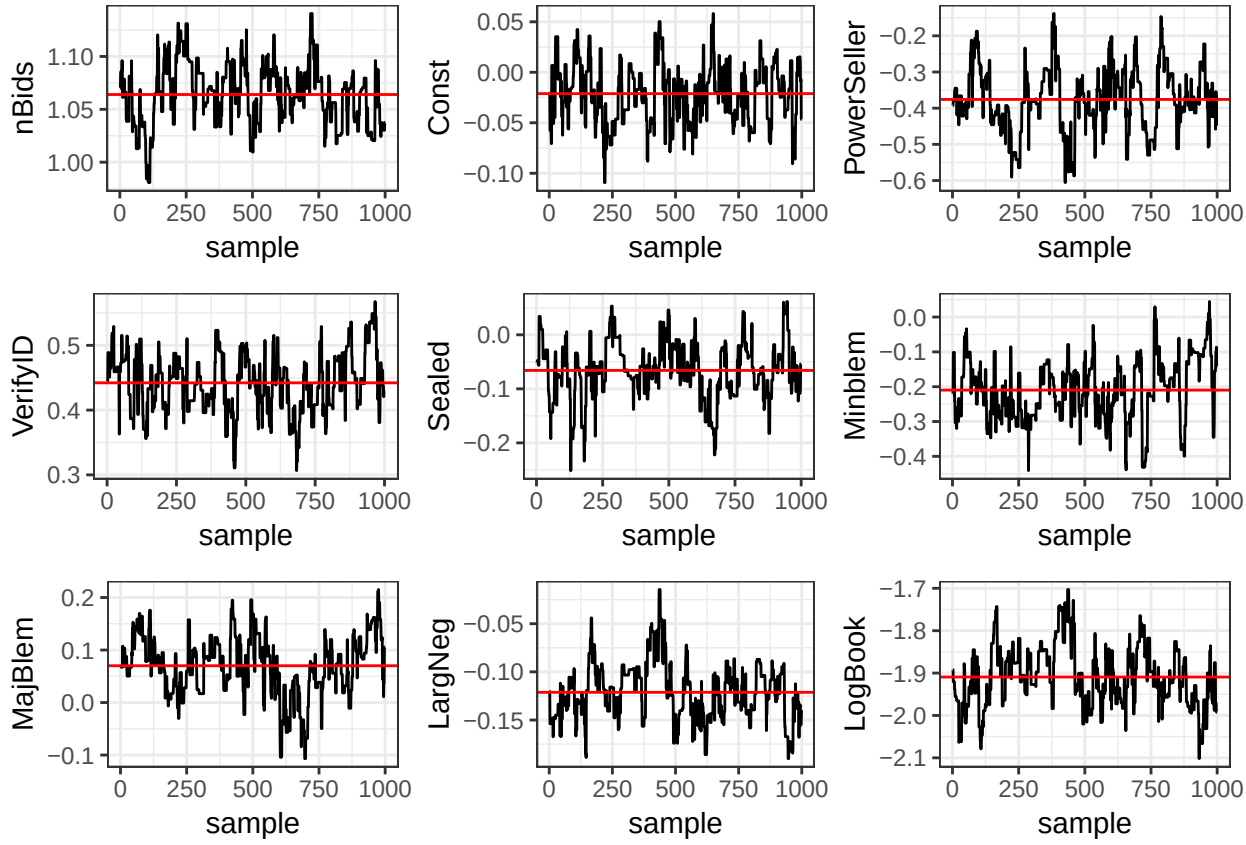
Using a value of 0.6 for c, an acceptance rate between 25 and 30 percent is obtained.

The convergence of each beta can be shown as follows:

```
library(gridExtra)
# Plotting convergence of beta.
  # Const.
  conv_Const = ggplot() +
    geom_line(aes(x = seq(1:nrow(res)),
                  y = res[,1])) +
    geom_hline(yintercept = mean(res[,1]),
               color = "red") +
    labs(x = "sample",
         y = colnames(data)[1]) +
    theme_bw()
  # PowerSeller
  conv_PowerSeller = ggplot() +
    geom_line(aes(x = seq(1:nrow(res)),
                  y = res[,2])) +
    geom_hline(yintercept = mean(res[,2]),
               color = "red") +
    labs(x = "sample",
         y = colnames(data)[2]) +
    theme_bw()
```

```r
# VerifyID
conv_VerifyID = ggplot() +
  geom_line(aes(x = seq(1:nrow(res)),
                y = res[,3])) +
  geom_hline(yintercept = mean(res[,3]),
             color = "red") +
  labs(x = "sample",
       y = colnames(data)[3]) +
  theme_bw()
# Sealed
conv_Sealed = ggplot() +
  geom_line(aes(x = seq(1:nrow(res)),
                y = res[,4])) +
  geom_hline(yintercept = mean(res[,4]),
             color = "red") +
  labs(x = "sample",
       y = colnames(data)[4]) +
  theme_bw()
# Minblem
conv_Minblem = ggplot() +
  geom_line(aes(x = seq(1:nrow(res)),
                y = res[,5])) +
  geom_hline(yintercept = mean(res[,5]),
             color = "red") +
  labs(x = "sample",
       y = colnames(data)[5]) +
  theme_bw()
# MajBlem
conv_MajBlem = ggplot() +
  geom_line(aes(x = seq(1:nrow(res)),
                y = res[,6])) +
  geom_hline(yintercept = mean(res[,6]),
             color = "red") +
  labs(x = "sample",
       y = colnames(data)[6]) +
  theme_bw()
# LargNeg
conv_LargNeg = ggplot() +
  geom_line(aes(x = seq(1:nrow(res)),
                y = res[,7])) +
  geom_hline(yintercept = mean(res[,7]),
             color = "red") +
  labs(x = "sample",
       y = colnames(data)[7]) +
  theme_bw()
# LogBook
conv_LogBook = ggplot() +
  geom_line(aes(x = seq(1:nrow(res)),
                y = res[,8])) +
  geom_hline(yintercept = mean(res[,8]),
             color = "red") +
  labs(x = "sample",
       y = colnames(data)[8]) +
```

```
    theme_bw()
  # MinBidShare
  conv_MinBidShare = ggplot() +
    geom_line(aes(x = seq(1:nrow(res)),
                  y = res[,9])) +
    geom_hline(yintercept = mean(res[,9]),
               color = "red") +
    labs(x = "sample",
         y = colnames(data)[9]) +
    theme_bw()

grid.arrange(conv_Const, conv_PowerSeller, conv_VerifyID, conv_Sealed,
             conv_Minblem, conv_MajBlem, conv_LargNeg, conv_LogBook,
             conv_MinBidShare, nrow = 3)
```



Appendix

Mattias Code for *1b)*

```
# Estimating a simple mixture of normals
# Author: Mattias Villani, IDA, Linkoping University. http://mattiasvillani.com

##########    BEGIN USER INPUT #################
# Data options
data = read.table("rainfall.dat", header = FALSE, sep = "", dec = ".")[,1]
x = as.matrix(data)
# Model options
nComp <- 2    # Number of mixture components

# Prior options
alpha <- 10*rep(1,nComp) # Dirichlet(alpha)
muPrior <- rep(0,nComp) # Prior mean of mu
tau2Prior <- rep(10,nComp) # Prior std of mu
sigma2_0 <- rep(var(x),nComp) # s20 (best guess of sigma2)
nu0 <- rep(4,nComp) # degrees of freedom for prior on sigma2

# MCMC options
nIter <- 100 # Number of Gibbs sampling draws

# Plotting options
plotFit <- TRUE
lineColors <- c("blue", "green", "magenta", 'yellow')
sleepTime <- 0.1 # Adding sleep time between iterations for plotting
################   END USER INPUT ###############

###### Defining a function that simulates from the
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 co
  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)
S <- t(rmultinom(nObs, size = 1 , prob = rep(1/nComp,nComp))) # nObs-by-nComp matrix with component all
mu <- quantile(x, probs = seq(0,1,length = nComp))
sigma2 <- rep(var(x),nComp)
probObsInComp <- rep(NA, nComp)

# Setting up the plot
xGrid <- seq(min(x)-1*apply(x,2,sd),max(x)+1*apply(x,2,sd),length = 100)
xGridMin <- min(xGrid)
xGridMax <- max(xGrid)
mixDensMean <- rep(0,length(xGrid))
effIterCount <- 0
ylim <- c(0,2*max(hist(x)$density))


for (k in 1:nIter){
  message(paste('Iteration number:',k))
  alloc <- S2alloc(S) # Just a function that converts between different representations of the group al
  nAlloc <- colSums(S)
  print(nAlloc)
  # Update components probabilities
  pi <- rDirichlet(alpha + nAlloc)

  # 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
    mu[j] <- rnorm(1, mean = muPost, sd = sqrt(tau2Post))
  }

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

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

  # Printing the fitted density against data histogram
  if (plotFit && (k%%1 ==0)){
    effIterCount <- effIterCount + 1
```

```
    hist(x, breaks = 20, freq = FALSE, xlim = c(xGridMin,xGridMax), main = paste("Iteration number",k),
    mixDens <- rep(0,length(xGrid))
    components <- c()
    for (j in 1:nComp){
      compDens <- dnorm(xGrid,mu[j],sd = sqrt(sigma2[j]))
      mixDens <- mixDens + pi[j]*compDens
      lines(xGrid, compDens, type = "l", lwd = 2, col = lineColors[j])
      components[j] <- paste("Component ",j)
    }
    mixDensMean <- ((effIterCount-1)*mixDensMean + mixDens)/effIterCount

    lines(xGrid, mixDens, type = "l", lty = 2, lwd = 3, col = 'red')
    legend("topleft", box.lty = 1, legend = c("Data histogram",components, 'Mixture'),
           col = c("black",lineColors[1:nComp], 'red'), lwd = 2)
    Sys.sleep(sleepTime)
  }

}

hist(x, breaks = 20, freq = FALSE, xlim = c(xGridMin,xGridMax), main = "Final fitted density")
lines(xGrid, mixDensMean, type = "l", lwd = 2, lty = 4, col = "red")
lines(xGrid, dnorm(xGrid, mean = mean(x), sd = apply(x,2,sd)), type = "l", lwd = 2, col = "blue")
legend("topright", box.lty = 1, legend = c("Data histogram","Mixture density","Normal density"), col=c(
```

Lab04

Assignment 1: Time series models in Stan

1a. Implementing and running function simulate from AR(1)-process.

Write a function in R that simulates data from the AR(1)-process

$$x_t = \mu + \phi(x_{t-1} - \mu) + \epsilon_t, \epsilon_t \sim N(0, \sigma^2)$$

for given values of μ, ϕ and σ^2. Start the process at $x_1 = \mu$ and then simulate values for x_t for t=2,3,...T and return the vector $x_{1:T}$ containing all time points. Use μ=10, σ^2=2 and T=200 and look at some different realizations (simulations) of $x_{1:T}$ for values of ϕ between -1 and 1 (this is the interval of ϕ where the AR(1)-process is stable). Include a plot of at least one realization in the report. What effect does the value of ϕ have on $x_{1:T}$?

Implementing function

First, the function which will be used to simulate from the AR(1)-process be implemented.

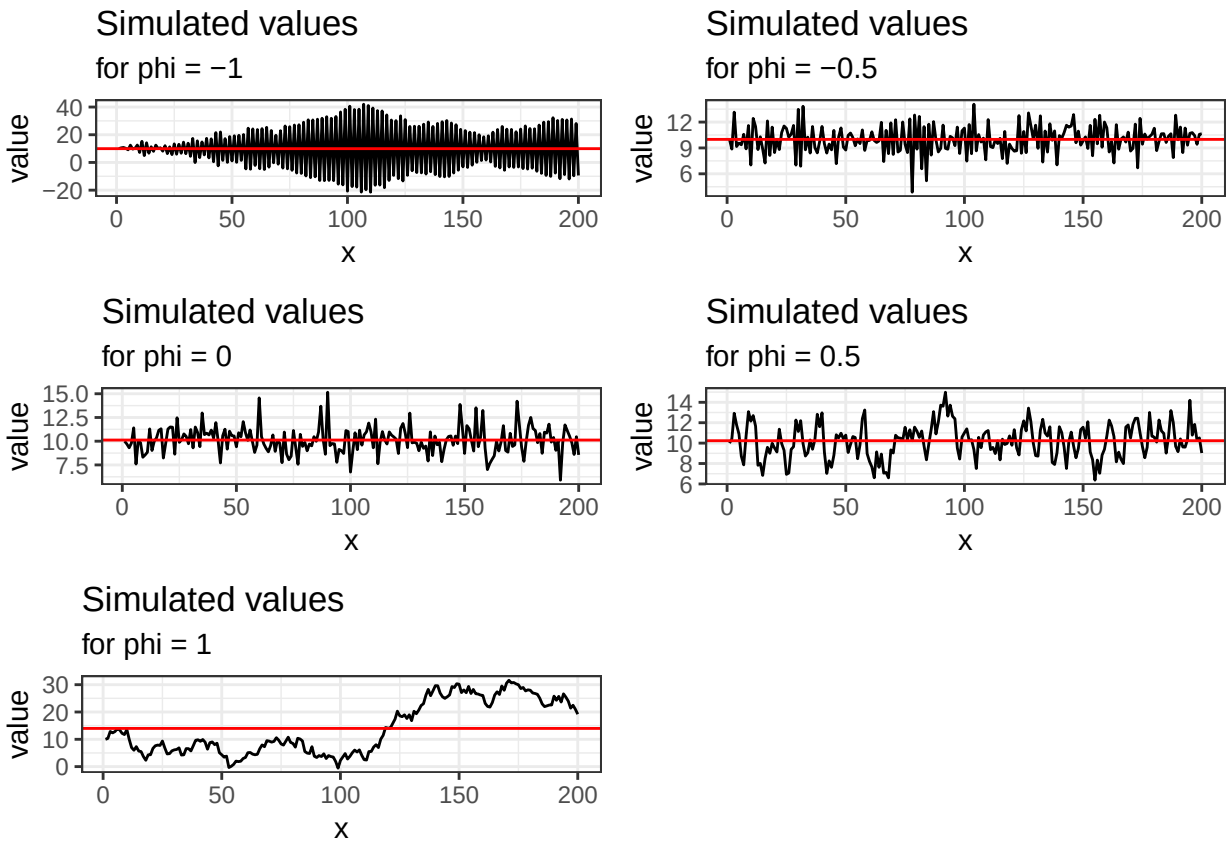
```
simulate_from_AR1_process = function(mu, phi, sigma_sq, T) {
  # Initializing vector to store simulations with mu as first value.
  res = mu
  # Initializing previous simulated value.
  sim_prev = mu
  # Simulating other 2:T values.
  for (t in 2:T) {
    # Simulating.
    sim_curr = mu+phi*(sim_prev-mu)+rnorm(n = 1, mean = 0, sd  = sqrt(sigma_sq))
    # Storing simulated value.
    res = c(res, sim_curr)
    # Setting simulated value to previous simulated value for next iteration.
    sim_prev = sim_curr
  }
  # Returning all stored simualted values.
  return(res)
}
```

Simulating from AR(1) for different values of ϕ

Using parameters $\mu = 10$, $\sigma^2 = 2$ and $T = 200$, we store the simulated results for different choices of ϕ which will be within the range from -1 to 1. To do so, we investigate the following different values for ϕ : -1, -0.5, 0, 0.5, 1.

```
# Setting up input parameters.
mu = 10
sigma_sq = 2
T = 200
phi_all = seq(from = -1, to = 1, by = 0.5)

# Simulating for differenct choices of phi.
sim_df = data.frame(matrix(NA, ncol = length(phi_all), nrow = T))
for (i in 1:length(phi_all)) {
  sim_df[, i] = simulate_from_AR1_process(mu = mu, phi = phi_all[i], sigma_sq = sigma_sq, T = T)
```

```
}

# Plotting results.
library(ggplot2)
## Creating function which will be create plot of same type for every different choice of phi.
plot_f = function(sim_idx) {
  res = ggplot(data = sim_df) +
    geom_line(aes(x = seq(1:T),
                  y = sim_df[, sim_idx])) +
    geom_hline(yintercept = mean(sim_df[, sim_idx]),
               color = "red") +
    theme_bw() +
    labs(title = "Simulated values",
         subtitle = paste0("for phi = ", as.character(phi_all[sim_idx])),
         x = "x",
         y = "value")
  return(res)
}
# Creating and storing plots for differenct choices of phi.
sim_list = list()
for (i in 1:length(phi_all)) {
  sim_list[[i]] = plot_f(i)
}
# Plotting.
library(gridExtra)
do.call(grid.arrange, c(sim_list, ncol = 2))
```



It can be seen that for ϕ-values of -1, -0.5, 0 and 0.5, the mean of the simulated values is close to μ. However, a different choice of ϕ leads to a different spread of the data. In contrast to this, choosing $\phi = 1$ also leads to a complete different mean of the data.

1b. Estimating parameters of AR(1)-model using Stan

Use your function from a) to simulate two AR(1)-processes, $x_{1:T}$ with $\phi = 0.3$ and $y_{1:T}$ with $\phi = 0.95$. Now, treat the values of μ, ϕ and σ^2 as unknown and estimate them using MCMC. Implement Stan-code that samples from the posterior of the three parameters, using suitable non-informative priors of your choice. [Hint: Look at the time-series models examples in the Stan reference manual, and note the different parameterization used here.]

i. Report the posterior mean of 95 percentage credible intervals and the number of effective posterior samples for the three inferred parameters for each of the simulated AR(1)-process. Are you able to estimate the true values?

ii. For each of the two data sets, evaluate the convergence of the samplers and plot the joint posterior of μ and ϕ. Comments?

Simulating from AR(1) with phi {0.3, 0.95}

First, we simulate two AR(1)-processes, $x_{1:T}$ with $\phi = 0.3$ and $y_{1:T}$ with $\phi = 0.95$. To do so, we use the same values for μ, σ^2 and T as before.

```
ar_phi.3 = simulate_from_AR1_process(mu = mu, phi = 0.3, sigma_sq = sigma_sq, T = T)
ar_phi.95 = simulate_from_AR1_process(mu = mu, phi = 0.95, sigma_sq = sigma_sq, T = T)
```

Implementing Stan model

The Stan-code that samples from the posterior of the three parameters, using suitable non-informative priors of your choice, will be implemented. In Stan, not specifying a prior is equivalent to specifying a uniform prior. Otherwise, we have to specify the prior within the model section. However, we stick to the uniform priors.

```
# Implementing Stan model.
library(rstan)
```

```
Loading required package: StanHeaders
```

```
rstan (Version 2.18.2, GitRev: 2e1f913d3ca3)
```

```
For execution on a local, multicore CPU with excess RAM we recommend calling
options(mc.cores = parallel::detectCores()).
To avoid recompilation of unchanged Stan programs, we recommend calling
rstan_options(auto_write = TRUE)
```

```
StanModel = '
data {
  int<lower=1> N; // N >= 1
  real x[N]; // of length N
}
parameters {
  real mu;
  real<lower=0> sigma_sq;
  real<lower=-1,upper=1> phi;
}
transformed parameters {
  real sigma;
  sigma = sqrt(sigma_sq);
}
```

```
model {
 // here we could specify other priors
 // like mu ~ normal()
 // or phi ~ uniform(-1, 1)
 for (n in 2:N)
   x[n] ~ normal(mu + phi * (x[n-1] - mu), sigma);
}'
```

Performing MCMC

```
# Setting up MCMC parameters.
burnin = 1000
niter = 2000

# Performing MCMC using StanModel.
stan_fit_phi.3 = stan(model_code = StanModel,
                      data = list(x = ar_phi.3,
                                  N = T),
                      control = list(adapt_delta = 0.99),
                      warmup = burnin,
                      iter = niter)
stan_fit_phi.95 = stan(model_code = StanModel,
                      data = list(x = ar_phi.95,
                                  N = T),
                      control = list(adapt_delta = 0.99),
                      warmup = burnin,
                      iter = niter)
```

Posterior means, 95% credible intervals and number of effective posterior samples

In the next step, we report the posterior means, 95% credible intervals and number of effective posterior samples.

```
knitr::kable(as.data.frame(summary(stan_fit_phi.3)$summary)[1:4, c(1, 4, 8, 9)],
             caption = "Results for phi = 0.3")
```

Table 7: Results for phi = 0.3

	mean	2.5%	97.5%	n_eff
mu	9.9817481	9.6789734	10.2848548	2417.103
sigma_sq	2.0810807	1.7096990	2.5221910	2877.518
phi	0.3359377	0.2058081	0.4683366	2900.704
sigma	1.4408219	1.3075546	1.5881407	2921.807

```
knitr::kable(as.data.frame(summary(stan_fit_phi.95)$summary)[1:4, c(1, 4, 8, 9)],
             caption = "Results for phi = 0.95")
```

Table 8: Results for phi = 0.95

	mean	2.5%	97.5%	n_eff
mu	-543.5513970	-34462.313002	1.069501e+04	39.488700
sigma_sq	1.7514498	1.429113	2.109771e+00	543.570843

	mean	2.5%	97.5%	n_eff
phi	0.9790409	0.900172	9.999991e-01	8.580531
sigma	1.3217673	1.195455	1.452505e+00	534.736712

```
# Other opportunities:
## Posterior mean.
### stan_post.means_phi.3 = get_posterior_mean(stan_fit_phi.3)

## Extracting params.
### stan_extr_phi.3 = extract(stan_fit_phi.3)

## Credible intervals.
### probs = c(0.025,0.975)
### apply(as.matrix(stan_extr_phi.3$mu), 2, quantile, probs=probs)
```

It can be seen that in both cases, ϕ and σ^2 are estimated pretty well. However, the estimate for μ is much better for the first fitted model with $\phi = 0.3$. For $\phi = 0.95$, the model estimate for μ is far away from the true μ.

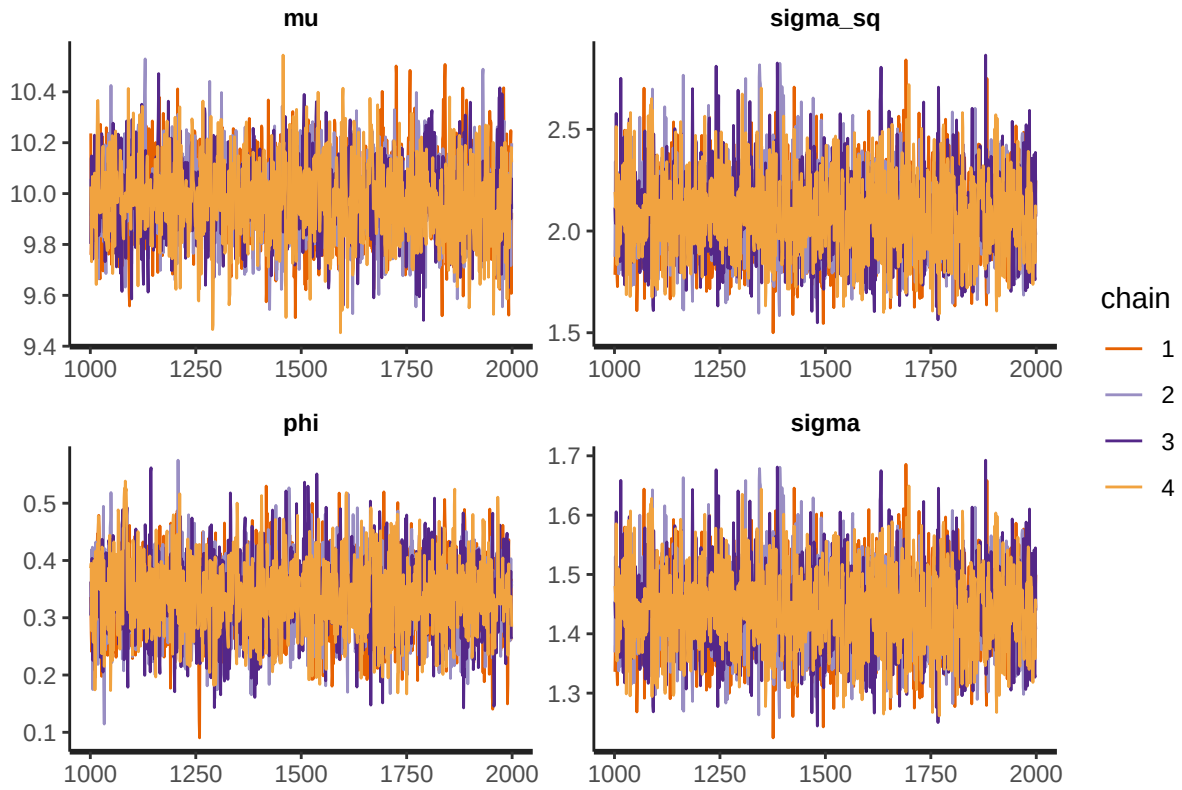
Convergence of the samplers

In the next step, we evaluate the convergence of the samplers.

```
# Plotting traceplots for every parameter.
cat("Traceplots for fitted model using dataset with phi = 0.3")
```

```
Traceplots for fitted model using dataset with phi = 0.3
```

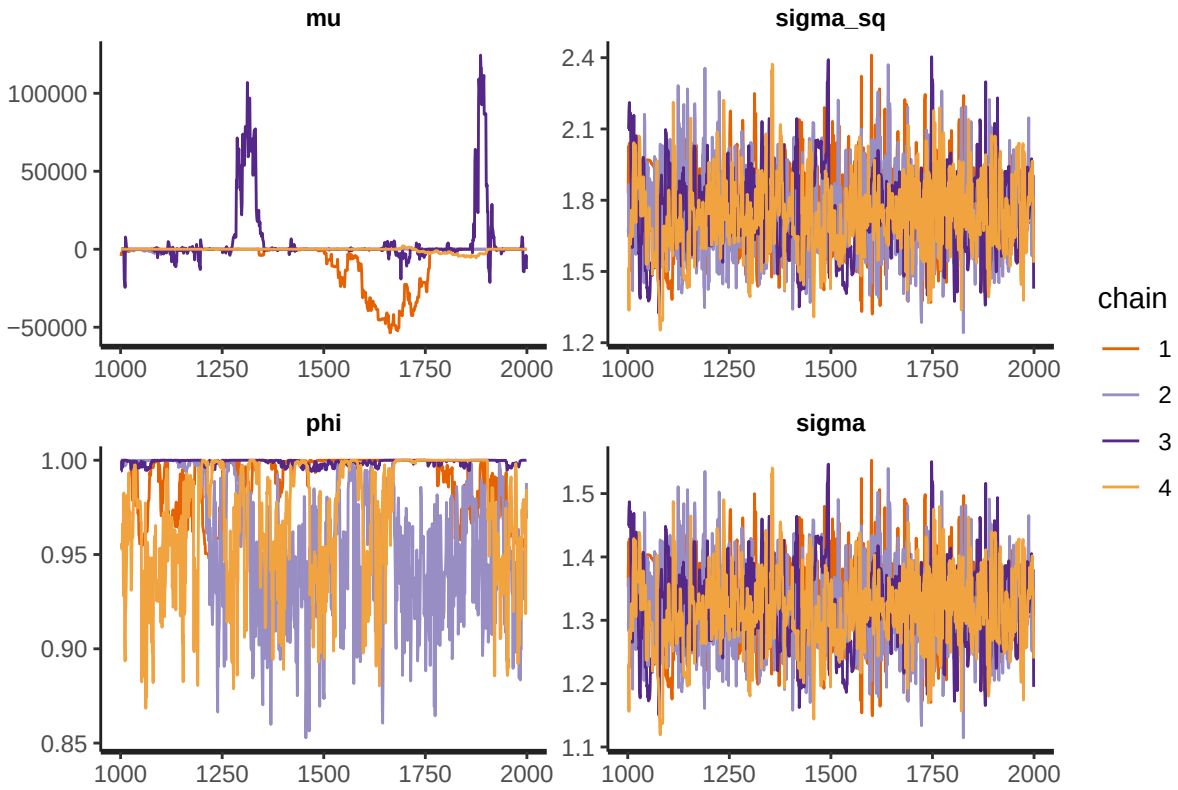
```
traceplot(stan_fit_phi.3)
```




```
cat("Traceplots for fitted model using dataset with phi = 0.95")
```

```
Traceplots for fitted model using dataset with phi = 0.95
```

```
traceplot(stan_fit_phi.95)
```

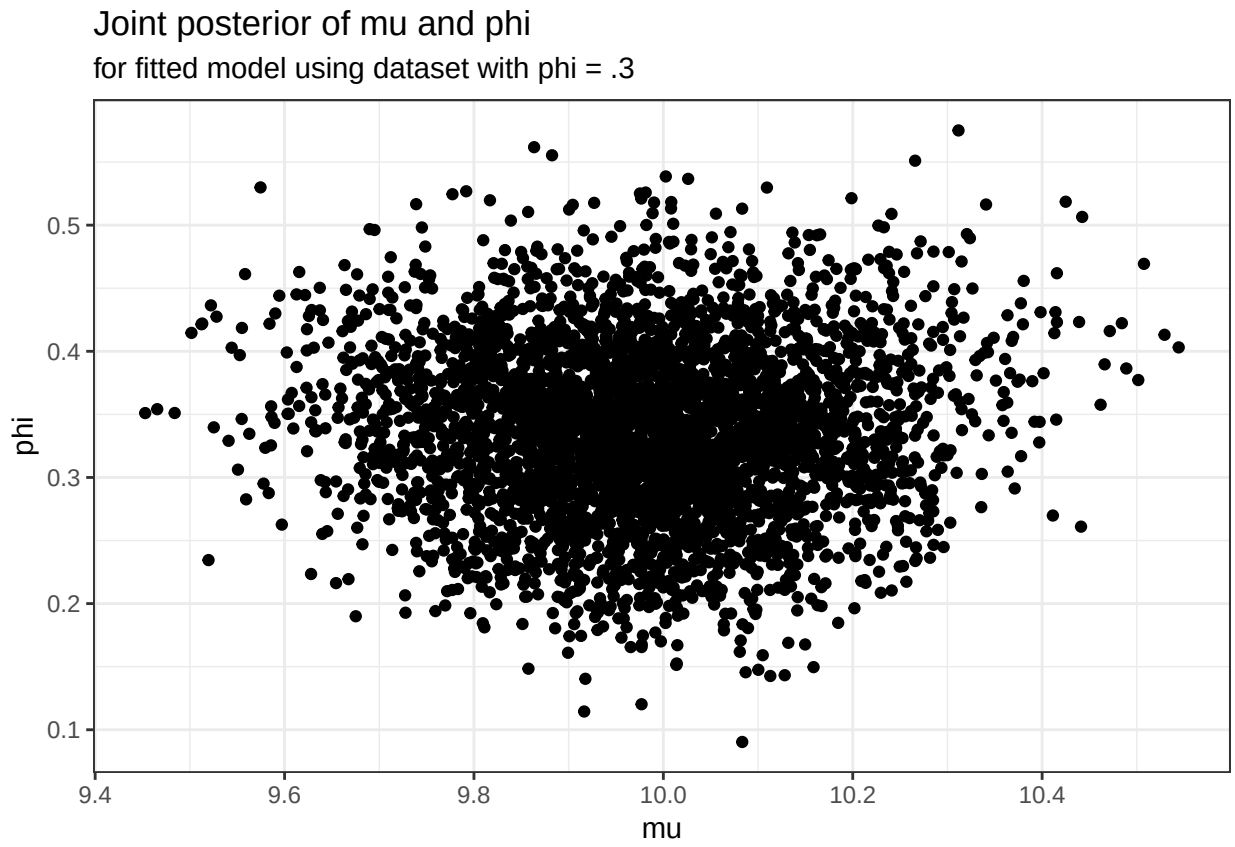



```
## In case that only one chain shall be printed: see lecture09, slide 11.
```

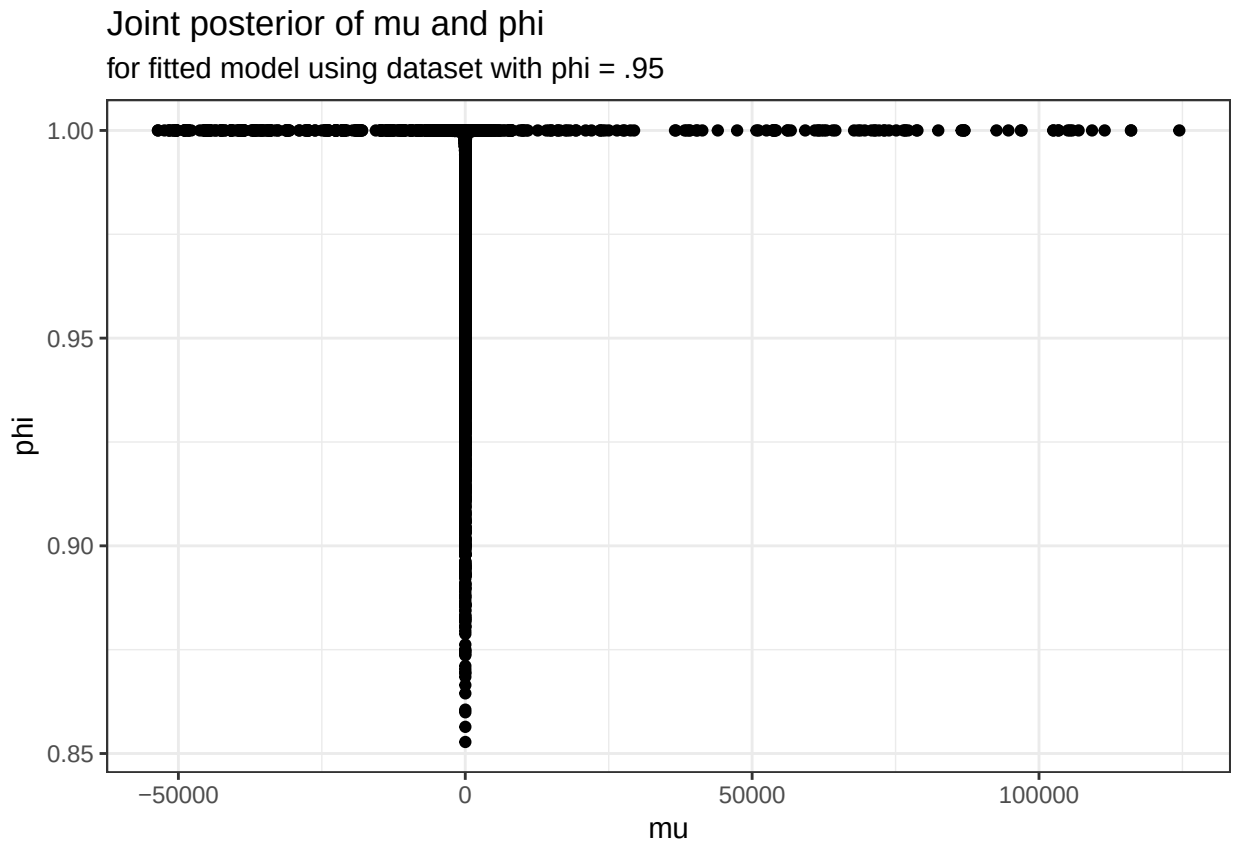
Joint posterior

Finally, we plot the joint posterior of μ and ϕ.

```
## Plotting joint posterior of mu and phi.
plot_joint_post = function(stan_fit) {
  plot_data = data.frame(mu = extract(stan_fit, pars = c("mu")), phi = extract(stan_fit, pars = c("phi"))
  res = ggplot(data = plot_data) +
    geom_point(aes(x = mu,
                   y = phi)) +
    theme_bw() +
    labs(title = "Joint posterior of mu and phi",
         subtitle = paste0("for fitted model using dataset with phi = ",
                           substr(as.character(substitute(stan_fit)), 13, 16)))
  print(res)
}
plot_joint_post(stan_fit_phi.3)
```




```
plot_joint_post(stan_fit_phi.95)
```


Joint posterior of mu and phi
for fitted model using dataset with phi = .95
1.00
0.95
0.90
0.85
phi
−50000
0
50000
100000
mu

1c. Estimating parameters of Poisson model conditioned on AR(1)-model using non-informative prior

The data campy.dat contain the number of cases of campylobacter infections in the north of the province Quebec (Canada) in four week intervals from January 1990 to the end of October 2000. It has 13 observations per year and 140 observations in total. Assume that the number of infections ct at each time point follows an independent Poisson distribution when conditioned on a latent AR(1)-process x_t, that is

$$c_t|x_t \sim Poisson(\exp(x_t))$$

where x_t is an AR(1)-process as in a). Implement and estimate the model in Stan, using suitable priors of your choice. Produce a plot that contains both the data and the posterior mean and 95 percent credible intervals for the latent intensity $\theta = \exp(x_t)$ over time. [Hint: Should x_t be seen as data or parameters?]

Reading data

First, we read the given data.

```
data = read.delim("campy.dat")[,1]
```

Implementing Stan model

Again, we are using uninformative priors for all parameters.

```
StanModel =
'
data {
  int<lower=0> N;
  int c[N]; // of length N
}
parameters {
  real mu;
  real<lower=0> sigma2;
  real phi;
  vector[N] x;
}
transformed parameters {
  real sigma;
  sigma = sqrt(sigma2);
}
model {
  for (n in 2:N)
  x[n] ~ normal(mu + phi * (x[n-1] - mu), sigma);
  for (n in 1:N)
  c[n] ~ poisson(exp(x[n]));
}
'
```

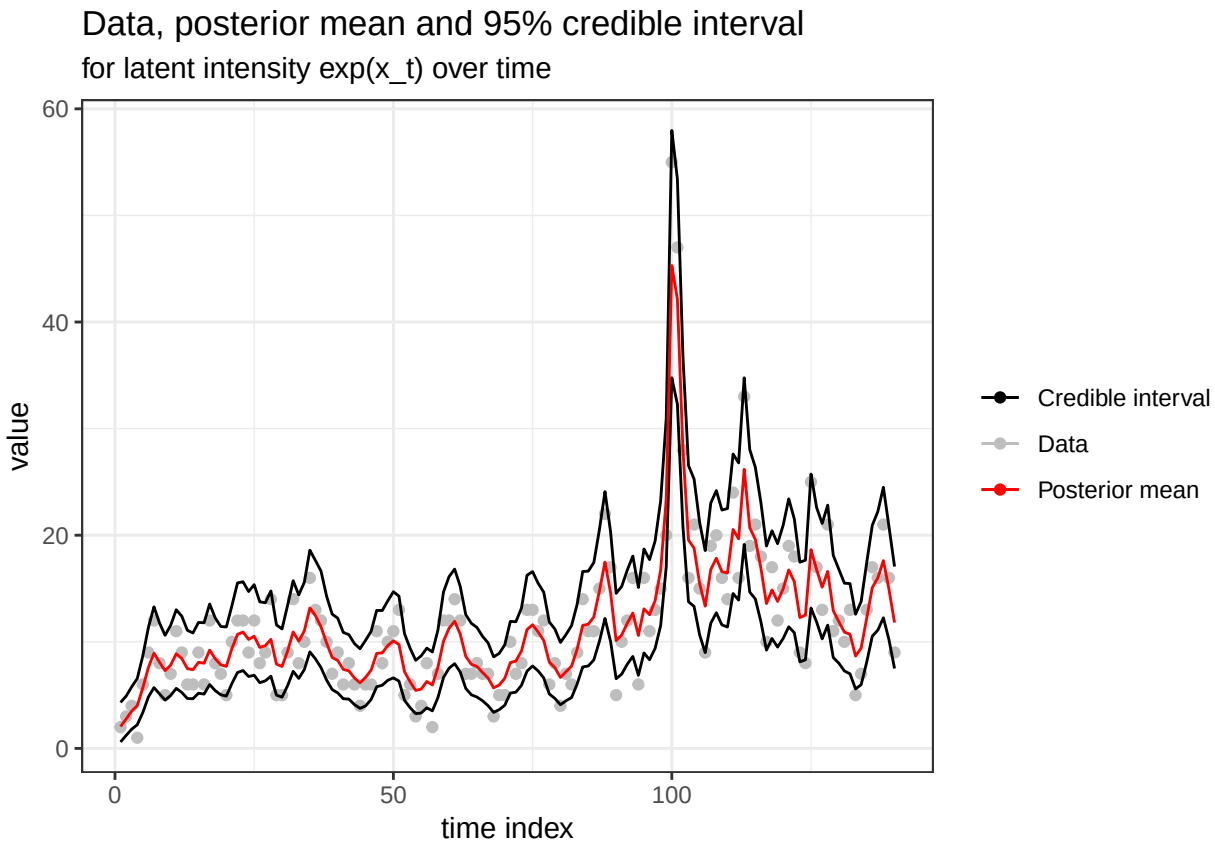
Performing MCMC

```
# Setting up MCMC parameters.
burnin = 1000
niter = 2000
```

```
# Performing MCMC using StanModel.
stan_fit_poisson = stan(model_code = StanModel,
                        data = list(c = data,
                                    N = length(data)),
                        control = list(adapt_delta = 0.99),
                        warmup = burnin,
                        iter = niter)
```

Plotting data, posterior mean and credible intervals for exp(x_t)

```
# Extracting CI and mean of exp(x_t). summary() not a solution, because it only refers to x_t.
x = exp(extract(stan_fit_poisson, pars = "x")$x)
x_lower = apply(x, 2, quantile, probs= 0.025)
x_upper = apply(x, 2, quantile, probs= 0.975)
x_mean = apply(x, 2, mean)

# If I want to do the procedure for x_t instead of exp(x_t):
## Getting summary for all x_t.
#params_excluded = which(rownames(summary(stan_fit_poisson)$summary) %in%
#                          c("mu", "sigma2", "phi", "sigma", "lp__"))
#summary_x = summary(stan_fit_poisson)$summary[-params_excluded, ]
## Extracting CI and mean of x_t.
#x_upper = summary_x[, "97.5%"]
#x_lower = summary_x[, "2.5%"]
#x_mean = summary_x[, "mean"]

# Plotting data, mean and CI of exp(x_t).
plot_data = data.frame(data = data, x_upper = x_upper, x_lower = x_lower, x_mean = x_mean)
ggplot(data = plot_data) +
  geom_point(aes(x = seq(1:length(data)),
                 y = data,
                 color = "Data")) +
  geom_line(aes(x = seq(1:length(data)),
                y = x_lower,
                color = "Credible interval")) +
  geom_line(aes(x = seq(1:length(data)),
                y = x_upper,
                color = "Credible interval")) +
  geom_line(aes(x = seq(1:length(data)),
                y = x_mean,
                color = "Posterior mean")) +
  theme_bw() +
  scale_color_manual(values = c("Data" = "grey",
                                "Credible interval" = "black",
                                "Posterior mean" = "red")) +
  labs(title = "Data, posterior mean and 95% credible interval",
       subtitle = "for latent intensity exp(x_t) over time",
       x = "time index",
       y = "value",
       colour = NULL)
```


Data, posterior mean and 95% credible interval
for latent intensity exp(x_t) over time
value
time index
0
20
40
60
0
50
100
Credible interval
Data
Posterior mean

1d. Estimating parameters of Poisson model conditioned on AR(1)-model using informative prior

Now, assume that we have a prior belief that the true underlying intensity θ varies more smoothly than the data suggests. Change the prior for σ^2 so that it becomes informative about that the AR(1)-process increments ϵ should be small. Re-estimate the model using Stan with the new prior and produce the same plot as in c). Has the posterior for θ changed?

Implementing Stan model

In contrast to 1c), we are using an informative prior for σ. Since the variance of the error term should be very small, we assume a normal prior for σ with a very small variance.

```
StanModel =
'
data {
  int<lower=0> N;
  int c[N]; // of length N
}
parameters {
  real mu;
  real<lower=0> sigma2;
  real phi;
  vector[N] x;
}
transformed parameters {
  real sigma;
  sigma = sqrt(sigma2);
}
model {
  sigma ~ normal(0, 0.02); //new prior for sigma
  for (n in 2:N)
  x[n] ~ normal(mu + phi * (x[n-1] - mu), sigma);
  for (n in 1:N)
  c[n] ~ poisson(exp(x[n]));
}
'
```

Performing MCMC

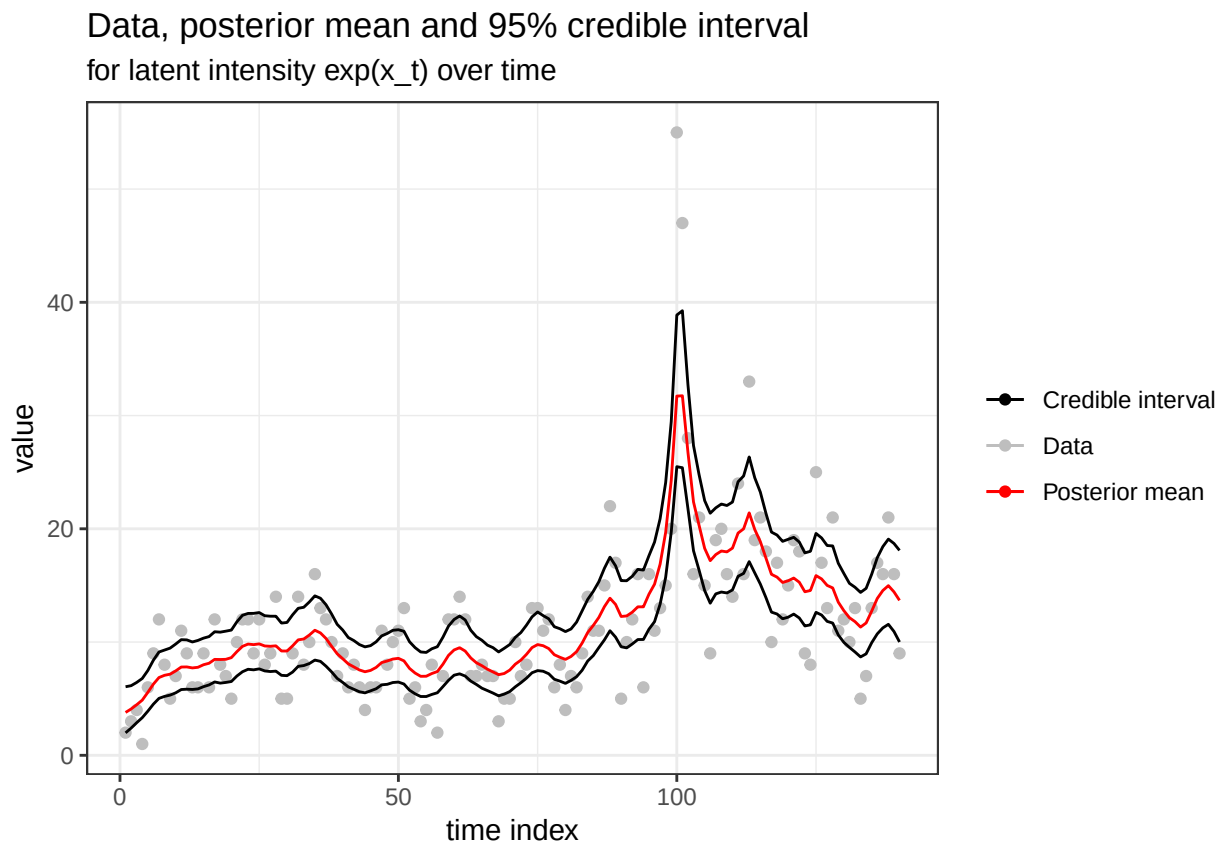
Then, we repeat the same process as in 1c).

```
# Setting up MCMC parameters.
burnin = 1000
niter = 2000

# Performing MCMC using StanModel.
stan_fit_poisson = stan(model_code = StanModel,
                        data = list(c = data,
                                    N = length(data)),
                        control = list(adapt_delta = 0.99),
                        warmup = burnin,
                        iter = niter)
```

Plotting data, posterior mean and credible intervals for exp(x_t)

Again, we repeat the same process as in 1c).

```
# Extracting CI and mean of exp(x_t). summary() not a solution, because it only refers to x_t.
x = exp(extract(stan_fit_poisson, pars = "x")$x)
x_lower = apply(x, 2, quantile, probs= 0.025)
x_upper = apply(x, 2, quantile, probs= 0.975)
x_mean = apply(x, 2, mean)

# If I want to do the procedure for x_t instead of exp(x_t):
## Getting summary for all x_t.
#params_excluded = which(rownames(summary(stan_fit_poisson)$summary) %in%
#                          c("mu", "sigma2", "phi", "sigma", "lp__"))
#summary_x = summary(stan_fit_poisson)$summary[-params_excluded, ]
## Extracting CI and mean of x_t.
#x_upper = summary_x[, "97.5%"]
#x_lower = summary_x[, "2.5%"]
#x_mean = summary_x[, "mean"]

# Plotting data, mean and CI of exp(x_t).
plot_data = data.frame(data = data, x_upper = x_upper, x_lower = x_lower, x_mean = x_mean)
ggplot(data = plot_data) +
  geom_point(aes(x = seq(1:length(data)),
                 y = data,
                 color = "Data")) +
  geom_line(aes(x = seq(1:length(data)),
                y = x_lower,
                color = "Credible interval")) +
  geom_line(aes(x = seq(1:length(data)),
                y = x_upper,
                color = "Credible interval")) +
  geom_line(aes(x = seq(1:length(data)),
                y = x_mean,
                color = "Posterior mean")) +
  theme_bw() +
  scale_color_manual(values = c("Data" = "grey",
                                "Credible interval" = "black",
                                "Posterior mean" = "red")) +
  labs(title = "Data, posterior mean and 95% credible interval",
       subtitle = "for latent intensity exp(x_t) over time",
       x = "time index",
       y = "value",
       colour = NULL)
```



It can be seen that the posterior of $exp(x_t)$ varies more smoothly than before.