Lab 4 - Computational Statistics (732A90)

February 21, 2018

Contents

1 Assignment 1: Computations with Metropolis–Hastings **1**
- 1.1 Task 1: Metropolis–Hastings Algorithm (log-normal as proposal distribution) . . . 1
- 1.2 Task 2: Metropolis–Hastings Algorithm (chi–square as proposal distribution) . . . 4
- 1.3 Task 3: Comparison of Task 1 and 2 . . . 6
- 1.4 Task 4: MCMS and Gelman–Rubin Method . . . 6
- 1.5 Task 5: Integral Estimation . . . 7
- 1.6 Task 6: Theoretical Integral . . . 8

2 Assignment 2: Gibbs sampling **9**
- 2.1 Task 1: Data Import and Visualization . . . 9
- 2.2 Task 2: Likelihood and prior for (random-walk) Bayesian model . . . 10
- 2.3 Task 3: Posterior for (random-walk) Bayesian model . . . 11
- 2.4 Task 4: Gibbs sampler . . . 13
- 2.5 Task 5: Trace plot and Conclusions . . . 15

3 Appendix **16**

1 Assignment 1: Computations with Metropolis–Hastings

1.1 Task 1: Metropolis–Hastings Algorithm (log-normal as proposal distribution)

We have the following pdf. We use the Metropolis–Hastings algorithm to generate samples from this distribution by using proposal distribution as log–normal $LN(X_t, 1)$.

$$f(x) \propto x^5 e^{-x}, x > 0$$

Note that the log-normal distribution $LN(\mu, \sigma)$ is not the log of the normal distribution but instead defined as follows (see https://de.wikipedia.org/wiki/Logarithmische_Normalverteilung):

$$LN(\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \frac{1}{x} \cdot \exp\left(-\frac{(ln(x) - ln(\mu))^2}{2\sigma^2}\right)$$

More specifically, the log-normal distribution is defined as follows in our case. Gladly, we can use the `rlnorm()` function in R, from the `stats` package to generate values from this distribution.

$$LN(X_t, 1) = \frac{1}{\sqrt{2\pi}} \cdot \frac{1}{x} \cdot \exp\left(-\frac{(ln(x) - ln(X_t))^2}{2}\right)$$

As a reference, the normal distribution is defined as follows:

$$N(\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

First, we implement the required functions: `pdf_sampling(x)`, `pdf_proposal(x, mu)`, `rng_proposal(Xt)`, `alpha(Xt, Y)`.

```
# Distribution proportionate to the distribution from which we want to sample
pdf_sampling = function(x) {

  x = ifelse(x <= 0, 0.00001, x)
  res = x^5 * exp(-x)
  return(res)

}

# Proposal distribution LN(Xt, 1)
pdf_proposal = function(x, mu) {

  res = (1 / sqrt(2 * pi)) * (1 / x) * exp(-(log(x) - log(mu))^2 / 2)
  # res = dlnorm(x, meanlog = mu, sdlog = 1) # SAME
  return(res)

}

# Random number generator corresponding to proposal distribution
rng_proposal = function(Xt){

  res = rlnorm(n = 1, meanlog = log(Xt), sdlog = 1)
  return(res)

}

# Acceptance probability alpha, required to check condition U < alpha(Xt, Y)
alpha = function(Xt, Y){

  num = pdf_sampling(Y) * pdf_proposal(Xt, Y)
  den = pdf_sampling(Xt) * pdf_proposal(Y, Xt)

  if (round(den, 8) == 0){    # Case 1: when denominator is almost zero
```

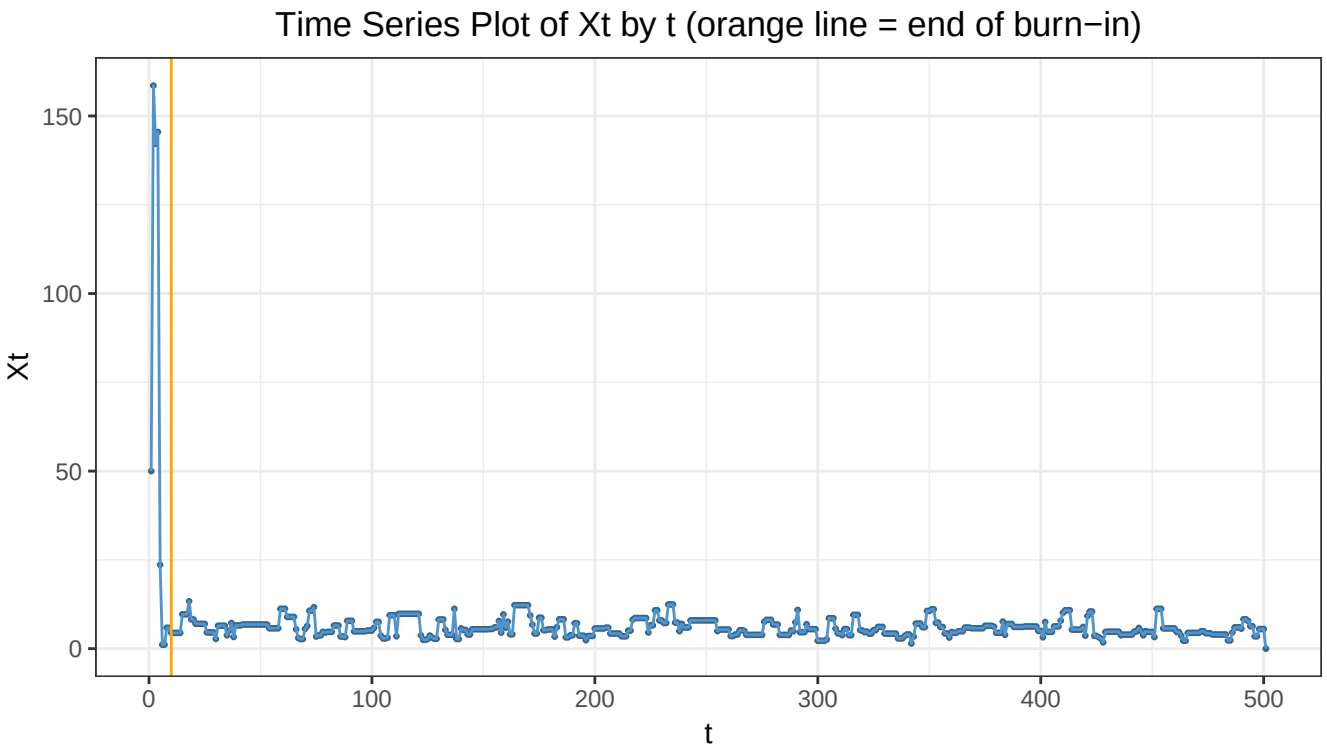
```
    return(1)
  } else {
    return(min(1, num / den)) # Case 2: all other cases
  }

}
```

Implementation of Metropolis Hastings Sampler

Then, we implement the Metropolis Hastings algorithm.

```
metropolis_hastings_sampler = function(init = 50, t_max = 500){

  # Initialize objects and paramters (We want X_0, X_1, ...X_t_max)
  X = numeric(length = (t_max + 1))
  X[1] = init
  t = 1

  # Conduct iterations of Metropolis Hastings
  while (t < t_max){

    U = runif(1)
    Y = rng_proposal(X[t])

    # a = ifelse(pdf_sampling(X[t]) <= 0, 0, alpha(X[t], Y))

    a = alpha(X[t], Y)
    if (U < a){
      X[t + 1] = Y
    } else {
      X[t + 1] = X[t]
    }

    t = t + 1
  }

  res = list(t = 1:(t_max + 1), Xt = X)

}
```

Plot of obtained chain as time series plot



Convergence of the chain

It seems like the chain does converge because from $t = 10$ onwards, the values circle around a range between ca. 0 and 15.

Note: without setting a seed, we also obtained (a few) results where the chain did not converge (before $t = 500$).

Size of burn-in period

The burn-in period seems to last until ca. $t = 10$ in this case as specified above.

Note: Without setting a seed, we also obtained (a few) results where the burn-in period lasted noticeably longer.

1.2 Task 2: Metropolis–Hastings Algorithm (chi–square as proposal distribution)

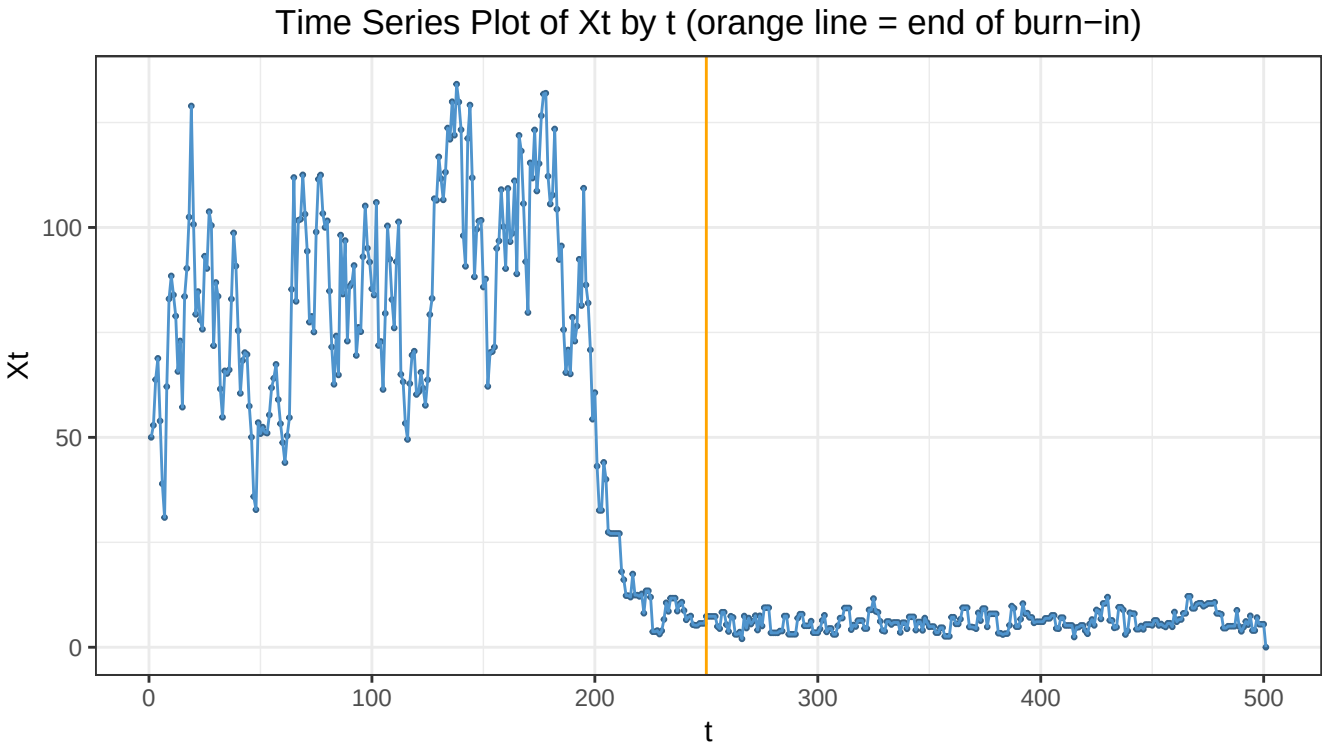
The only changes to be made are the following:

- convert `pdf_proposal()` to the pdf of the chi-square distribution
- generate $Y \sim \chi^2(\lfloor X_t + 1 \rfloor)$ instead of $Y \sim LN(X_t, 1)$ in the Metropolis Hastings algorithm

```
# Proposal distribution LN(Xt, 1)
pdf_proposal = function(x, Xt) {

  res = dchisq(x, df = floor(Xt + 1))
```

```r
  return(res)

}

# Random number generator corresponding to proposal distribution
rng_proposal = function(Xt){

  res = rchisq(n = 1, df = floor(Xt + 1))
  return(res)

}
```

Plot of obtained chain as time series plot



Convergence of the chain

It seems like the chain does converge because from $t = 250$ onwards, the values circle around a range between ca. 0 and 15.

Note: Without setting a seed, we also obtained (a few) results where the chain did not converge (before $t = 500$). Also, many other results looked similar to the previous plot generated with the log-normal distribution.

Size of burn-in period

The burn-in period seems to last until ca. $t = 250$ in this case as specified above.

Note: Without setting a seed, we also obtained (a few) results where the burn-in period lasted even longer. However, for most results, the burn-in period was noticeably shorter.

1.3 Task 3: Comparison of Task 1 and 2

After running the Metropolis–Hastings algorithm with both, the settings from task 1 and task 2 several times, it seems like the difference between using the log-normal and the chi-square distribution as proposal distributions is not as large as it seems based on the above plots (where the seed `12345` was used).

With seed `12345`

- the algorithm convergences at some point for both proposal distributions
- convergence takes more time for the chi-square proposal distribution
- after convergence, the results look very similar for both proposal distributions

In general (repeated runs without seed)

- there were cases with the algorithm not converging (before $t = 500$) for both both proposal distributions
- most of the time, the convergence was reached rather early (at around $t = 50$) for both proposal distributions

1.4 Task 4: MCMS and Gelman–Rubin Method

Here, we generate 10 MCMC sequences using the generator from task 2 and starting points 1, 2, ..., or 10. Subsequently, we use the Gelman–Rubin method to analyze convergence of these sequences.

First, we create a matrix M with dimensions $[k \times n] = [10 \times t_{max}]$ that holds an MCMC sequence in every row:

```
X_0 = 1:10
t_max = 1000
M = matrix(nrow = length(X_0), ncol = t_max)

for (i in 1:10){

  res = metropolis_hastings_sampler(init = i, t_max = t_max)
  M[i, ] = res$Xt[-1] # we remove the initial value

}
```

Subsequently, we compute the between- and within-sequence variances, B and W respectively. B and W are then used to compute the overall variance estimate $Var[v]$ which in turn is then used to compute the Gelman-Rubin factor $\sqrt{R}$.

$$B = \frac{n}{k-1}\sum_{i=1}^{k}(\bar{v}_{i\cdot} - \bar{v}_{\cdot\cdot})^2$$

$$W = \sum_{i=1}^{k}\frac{s_i^2}{k}$$

$$s_i^2 = \sum_{j=1}^{n}\frac{(\bar{v}_{ij} - \bar{v}_{i\cdot})^2}{n-1}$$

$$\hat{Var}[v] = \frac{n-1}{n}W + \frac{1}{n}B$$

$$Gelman - Rubin = \sqrt{R} = \sqrt{\frac{\hat{Var}[v]}{W}}$$

Gelman Rubin Factor (manual computation)

The computations are as follows.

```
n = ncol(M)
k = nrow(M)
v_bar = mean(M)

s_2 = colSums((M - rowMeans(M))^2 / (n - 1)) # vector of length k (with index i)
B = n / (k - 1) * sum((rowMeans(M) - v_bar)^2) # scalar
W = sum(s_2 / k) # scalar

var_hat = (n - 1) / n * W + (1 / n) * B
gelman_rubin_factor = sqrt(var_hat / W)

cat("Gelman Rubin Factor: ", gelman_rubin_factor, "\n")
```

```
## Gelman Rubin Factor:  1.002096
```

According to lecture 4, values much larger than 1 indicate lack of convergence. Since our result is very close to 1, we can conclude that the results did converge.

Gelman Rubin Factor (coda::gelman.diag())

If we use the `coda::gelman.diag()` function, we obtain very similar results than from the manual computation. This is good because it indicates that our manual computation was correct.

```
##      Point est. Upper C.I.
## [1,]   1.012334   1.025755
```

1.5 Task 5: Integral Estimation

Here, we estimate the following integral using the generators from task 1 and 2. Since the integral below represents the mean of the distribution from which we want to sample, we compute the sample mean of the obtained MCMC sequences from task 1 and 2 to estimate the integral.

Note: At this point, we do not now f(x). Instead, we only know a function that is proportionate to f(x), since we are given $f(x) \propto x^5 e^{-x}, x > 0$.

$$\int_0^\infty x f(x) dx$$

Estimation with Samples from Task 1 and 2

```
## Sample mean of MCMC sequence (task 1): 5.93119
```

```
## Sample mean of MCMC sequence (task 2): 6.173173
```

1.6 Task 6: Theoretical Integral

We are given that the distribution generated is in fact a gamma distribution. We know that the integral represents the mean of the distribution, therefore, it should evaluate to the mean of the gamma distribution.

The gamma distribution is defined as follows:

$$f(y) = \left[\frac{1}{\Gamma(\alpha)\beta^\alpha}\right] y^{\alpha-1} e^{-y/\beta}$$

where,

$$\Gamma(\alpha) = \int_0^\infty x^{\alpha-1} e^{-\alpha} dx = (\alpha - 1)!$$

Expected integral value

The mean of the Gamma distribution is defined as $\alpha \cdot \beta$. In our case, we have $\alpha = 6$ and $\beta = 1$. Therefore, the integral, which represents the mean of the distribution, evaluates to $\mu = 6 \cdot 1 = 6$.

Constant of proportionality c

Note that we have $f(x) \propto x^5 e^{-x}, x > 0$, **not** $f(x) = x^5 e^{-x}, x > 0$. Computing an integral over a function $x^5 e^{-x}$ would issue a different result, namely $\Gamma(7) = (7-1)! = 720$ because this function is not equal to $f(x)$ but only proportionate to $f(x)$. We can conclude that the constant of proportionality must be $1/120$ because we need to divide 720 by 120 in order to get to our desired value of 6.

Alternatively, we can compute the constant using the factor from the gamma distribution (above):

$$c = \left[\frac{1}{\Gamma(\alpha) \cdot \beta^\alpha}\right] = \left[\frac{1}{\Gamma(6) \cdot 1^6}\right] = \left[\frac{1}{120}\right]$$

Comparison with observed integral value

The theoretical value of 6 matches well with the observed values we obtained with the samples from task 1 and 2, which were ca. 5.93 and 6.17 respectively.

References:

- https://de.wikipedia.org/wiki/Gammafunktion
- https://www.britannica.com/science/gamma-function

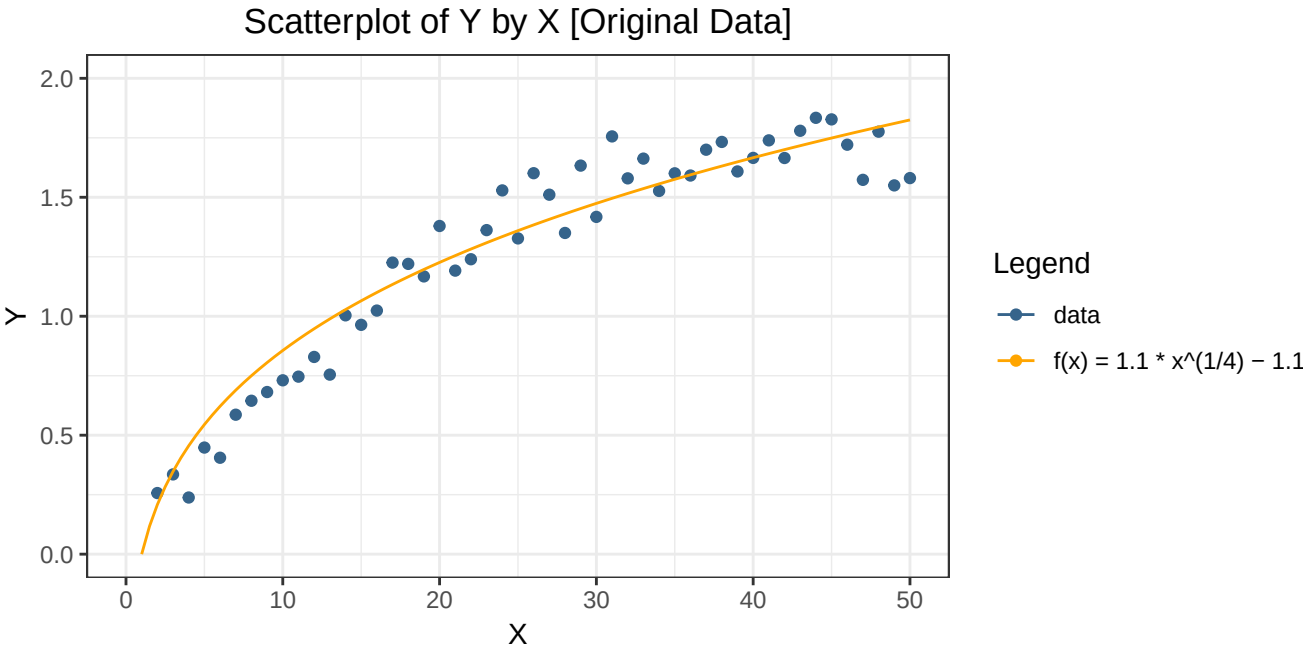
2 Assignment 2: Gibbs sampling

We are given a data set with 2 variables about the measurement of the concentration of a ceratain chemical:

- `X`: day of the measurement
- `Y`: measured concentration of the chemical

2.1 Task 1: Data Import and Visualization

First, we import the data and plot the dependence of Y on X. We can see that a linear line would not fit the data very well. The orange line seems to represent the data quite well and it is a non-linear function ($f(x) = 1.1 \cdot X^{\frac{1}{4}} - 1.1$).

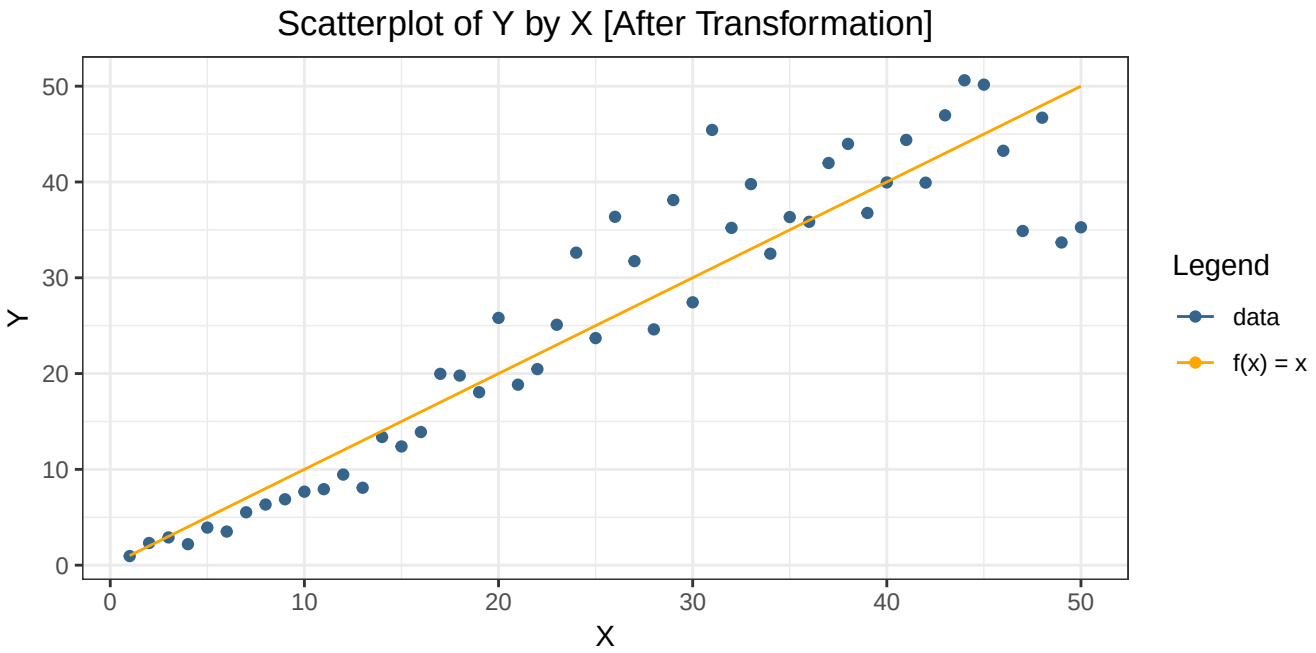


Option 1: Model

We could transform the data so that the relationship between X and Y is linear. E.g. we can solve the equation for the orange line in the plot above for x. The resulting expression is our new Y. After the transformation, we can fit a linear model.

$$Y = 1.1 \cdot X^{\frac{1}{4}} - 1.1$$
$$X = \left(\frac{Y + 1.1}{1.1}\right)^4$$

```r
# Transforming Y so that relationship between X and Y is linear
df$Y = ((Y + 1.1)/(1.1))^4
```



Option 2: Model

We could consider other non-linear prediction models (e.g. neural networks).

2.2 Task 2: Likelihood and prior for (random-walk) Bayesian model

We are given that the following (random-walk) Bayesian model (n = number of observations, $\boldsymbol{\mu} = (\mu_1, ..., \mu_n)$ are unknown parameters).

$$Y_i \sim N(\mu_i, \sigma^2 = 0.2), i = 1, ..., n$$

where the prior is:

$$P(\mu_1) = 1$$

$P(\mu_{i+1}|\mu_i) = N(\mu_i, \sigma^2 = 0.2), i = 1, ..., n - 1\$$

Formula showing the likelihood

$$\begin{aligned} P(\boldsymbol{Y}|\boldsymbol{\mu}) &= \prod_{i=1}^{n} N(\mu_i, \sigma^2 = 0.2) \\ &= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(y_i-\mu_i)^2} \\ &= \prod_{i=1}^{n} \frac{1}{\sqrt{0.4\pi}} e^{-\frac{1}{0.4}(y_i-\mu_i)^2} \\ &= \frac{1}{(\sqrt{0.4\pi})^n} e^{-\frac{1}{0.4}\sum_{i=1}^{n}(y_i-\mu_i)^2} \end{aligned}$$

Formula showing the prior

$$\begin{aligned} P(\boldsymbol{\mu}) &= \prod_{i=2}^{n} N(\mu_{i-1}, \sigma^2 = 0.2) \\ &= \frac{1}{(\sqrt{0.4\pi})^{(n-1)}} e^{-\frac{1}{0.4}\sum_{i=2}^{n}(\mu_i-\mu_{i-1})^2} \end{aligned}$$

2.3 Task 3: Posterior for (random-walk) Bayesian model

Here, we use the Bayes' Theorem to get the posterior up to a constant of proportionality, and then find out the distributions of $(\mu_i|\boldsymbol{\mu_{-i}},\boldsymbol{Y})$ where $\boldsymbol{\mu_{-i}}$ is a vector containing all μ values except of μ_i.

Bayes Theorem

Recall, that the Bayes Theorem is defined as follows:

$$P(\theta|D) = \frac{P(D|\theta) \cdot P(\theta)}{P(D)}$$

If we consider the posterior up to a constant of proportionality, we get:

$$P(\theta|D) \propto P(D|\theta) \cdot P(\theta)$$

Posterior

In our case, the posterior up to a constant of proportionality is:

$$\begin{aligned} P(\boldsymbol{\mu}|\boldsymbol{Y}) &\propto P(\boldsymbol{Y}|\boldsymbol{\mu}) \cdot P(\boldsymbol{\mu}) \\ &\propto \left[\prod_{i=1}^{n} N(\mu_i, \sigma^2 = 0.2)\right] \cdot \left[\prod_{i=2}^{n} N(\mu_{i-1}, \sigma^2 = 0.2)\right] \\ &\propto \left[\frac{1}{(\sqrt{0.4\pi})^n} e^{-\frac{1}{0.4}\sum_{i=1}^{n}(y_i-\mu_i)^2}\right] \cdot \left[\frac{1}{(\sqrt{0.4\pi})^{(n-1)}} e^{-\frac{1}{0.4}\sum_{i=2}^{n}(\mu_i-\mu_{i-1})^2}\right] \\ &\propto \left[e^{-\frac{1}{0.4}\sum_{i=1}^{n}(y_i-\mu_i)^2}\right] \cdot \left[e^{-\frac{1}{0.4}\sum_{i=2}^{n}(\mu_i-\mu_{i-1})^2}\right] \\ &\propto e^{-\frac{1}{0.4}\cdot\left(\sum_{i=1}^{n}(y_i-\mu_i)^2+\sum_{i=2}^{n}(\mu_i-\mu_{i-1})^2\right)} \end{aligned}$$

Now, we can use hint B from the instructions. But first, we re-write the above expression so that the use of the hint gets more clear:

$$P(\boldsymbol{\mu}|\boldsymbol{Y}) \propto \exp\left(-\frac{1}{0.4} \cdot \left[(y_1-\mu_1)^2 + (y_2-\mu_2)^2 + ... + (y_{n-1}-\mu_{n-1})^2 + (y_n-\mu_n)^2\right]\right)$$
$$\cdot \exp\left(-\frac{1}{0.4} \cdot \left[(\mu_2-\mu_1)^2 + (\mu_3-\mu_2)^2 + ... + (\mu_n-\mu_{n-1})^2\right]\right)$$

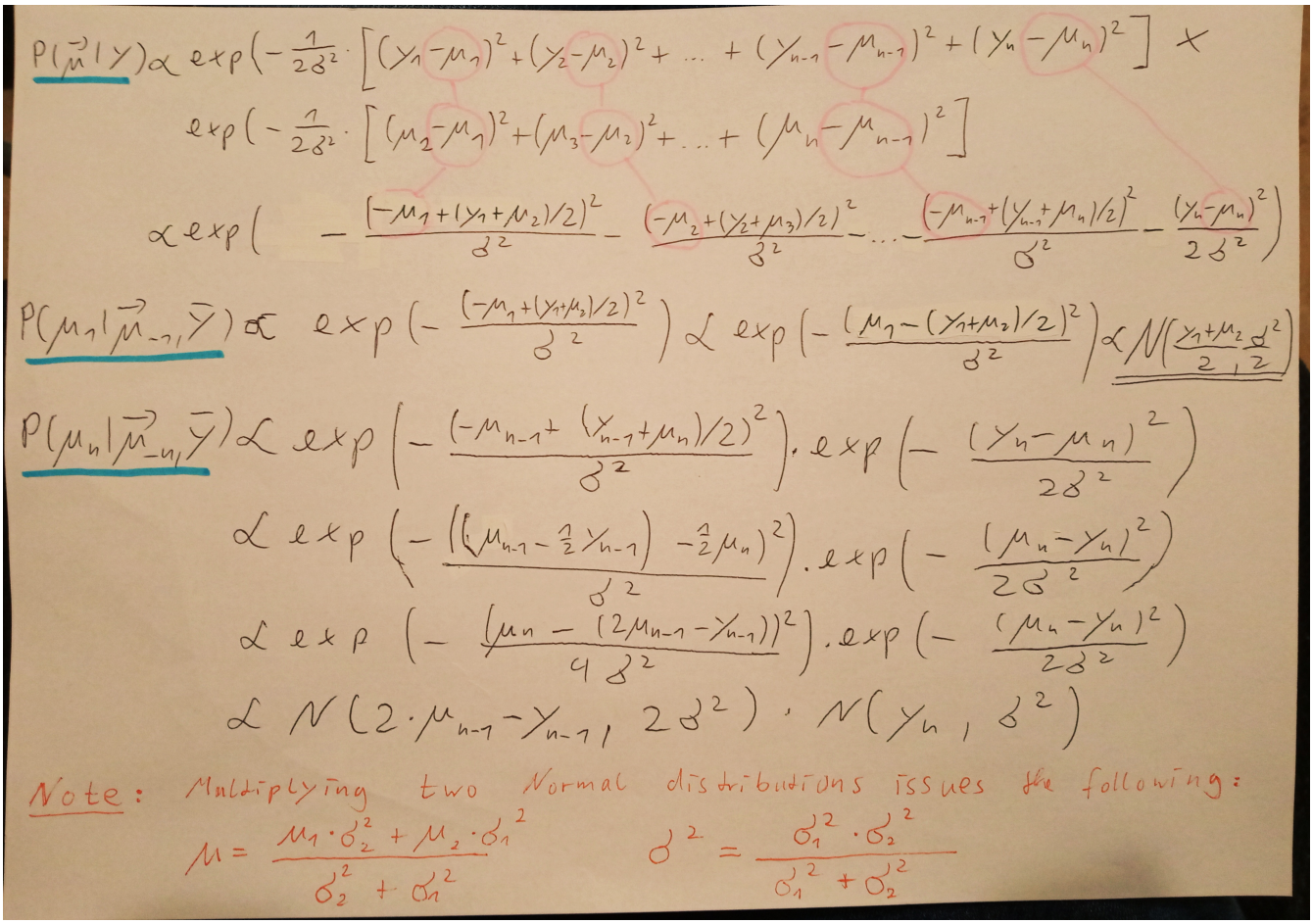
Subsequently, the distributions $(\mu_1|\boldsymbol{\mu_{-1}},\boldsymbol{Y})$, $(\mu_n|\boldsymbol{\mu_{-n}},\boldsymbol{Y})$, and $(\mu_i|\boldsymbol{\mu_{-i}},\boldsymbol{Y})$ are derived. In the Gibbs sampling procedure, we sample $\mu_{t+1,i}$ from these distributions. The final formulas are shown below and the derivations follow in form of pictures afterwards.

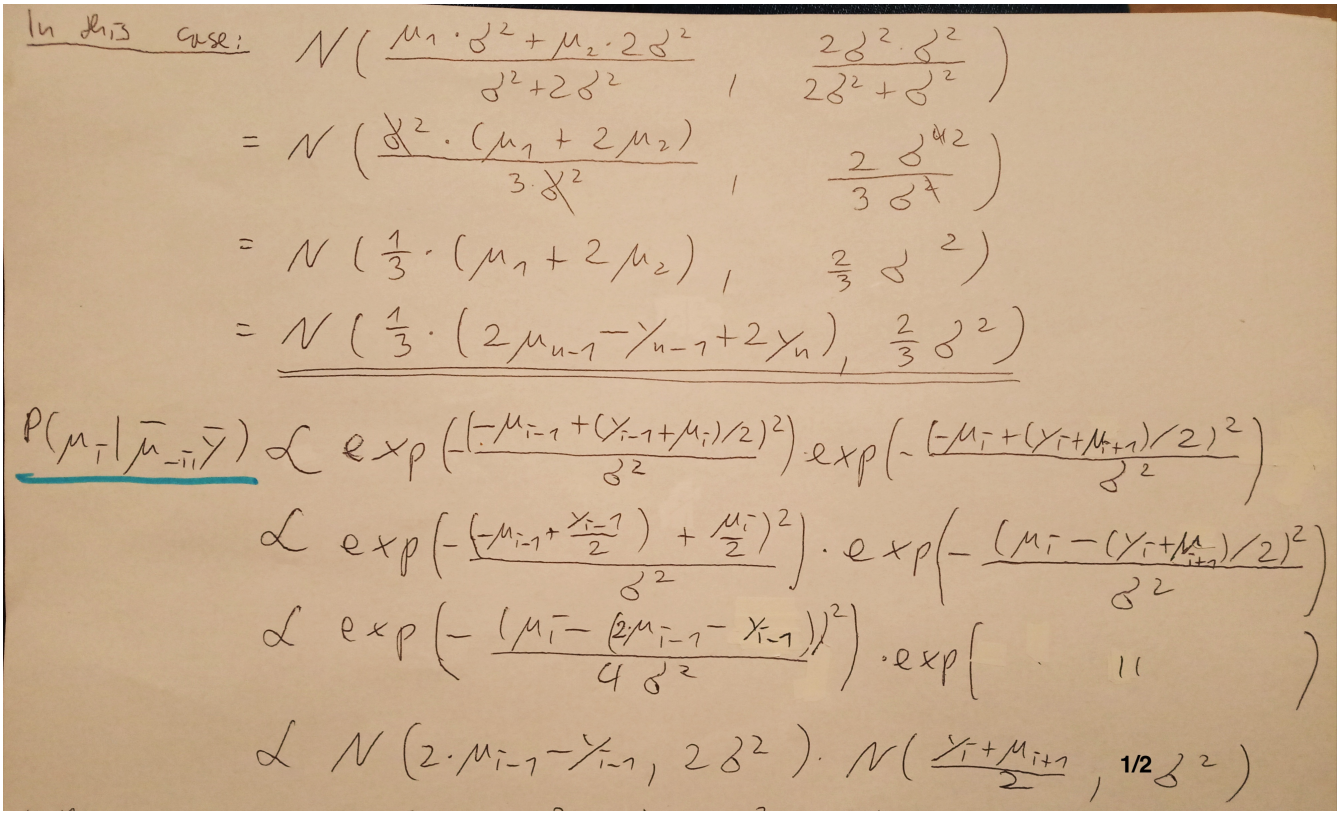
Distributions of $(\mu_i|\boldsymbol{\mu_{-i}},\boldsymbol{Y})$

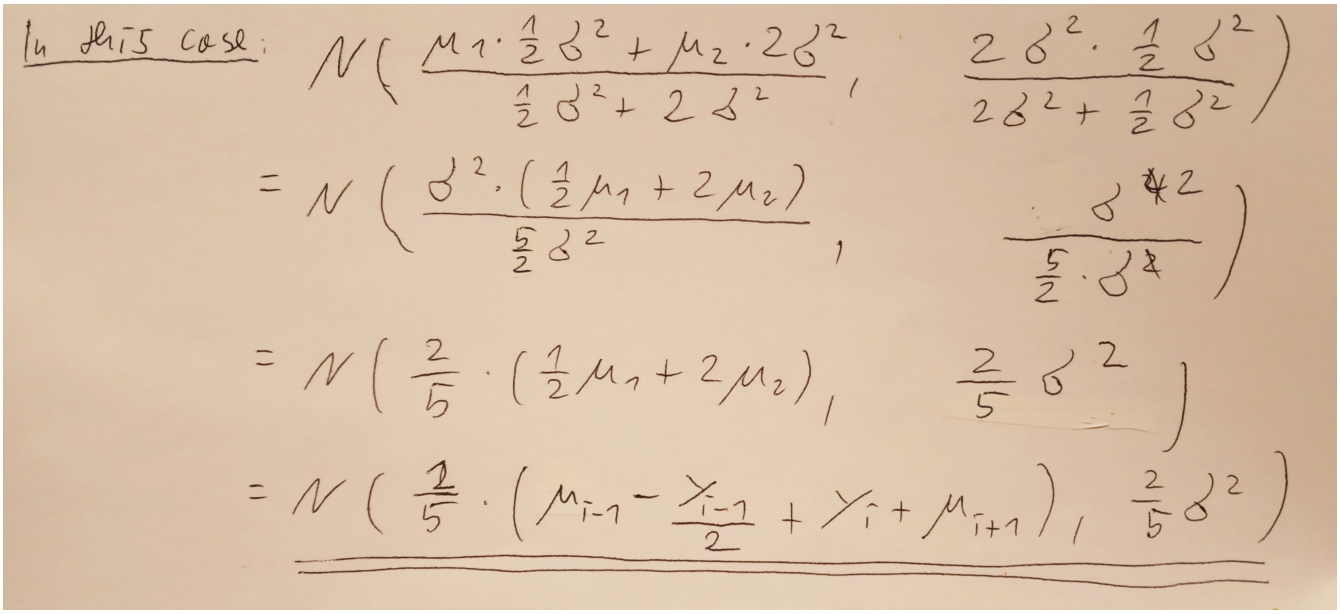
$$P(\mu_1|\boldsymbol{\mu_{-1}},\boldsymbol{Y}) = N\left(\frac{y_1+\mu_2}{2}, \frac{\sigma^2}{2}\right)$$

$$(\mu_n|\boldsymbol{\mu_{-n}},\boldsymbol{Y}) = N\left(\frac{1}{3}\left(2\mu_{n-1} - y_{n-1} + 2y_n\right), \frac{2}{3}\sigma^2\right)$$

$$(\mu_i|\boldsymbol{\mu_{-i}},\boldsymbol{Y}) = N\left(\frac{2}{5}(\mu_{i-1} - \frac{y_{i-1}}{2} + y_i + \mu_{i+1}), \frac{2}{5}\sigma^2\right)$$



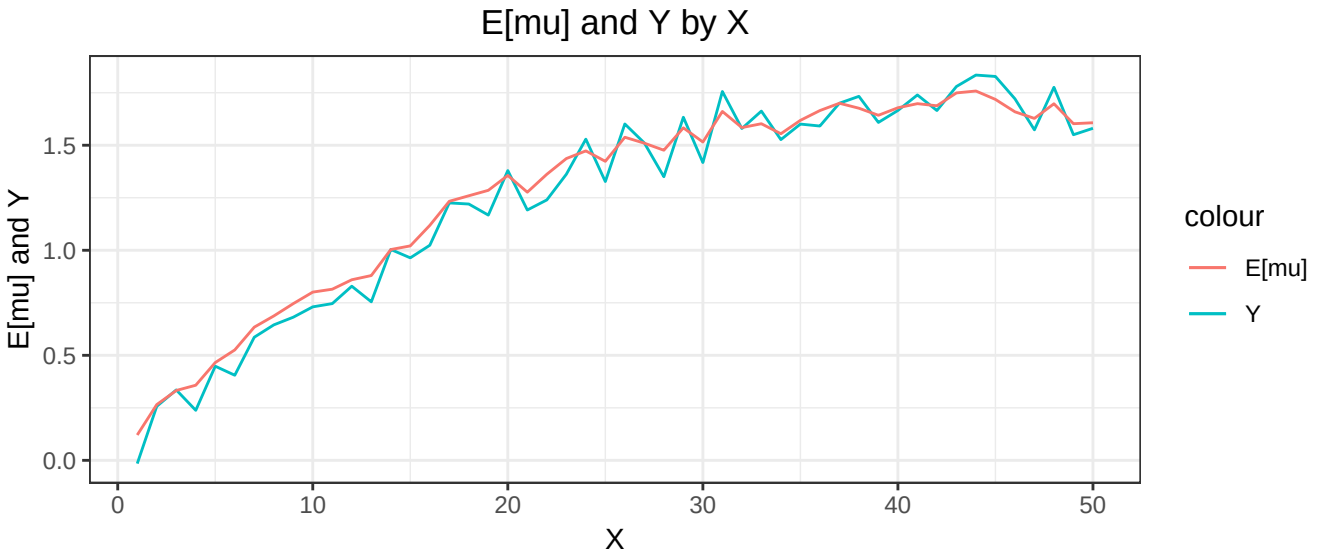




2.4 Task 4: Gibbs sampler

Here, we use the distributions derived in task 3 to implement a Gibbs sampler that uses $\mu_0 = (0, ..., 0)$ as a starting point. We run the Gibbs sampler to obtain 1000 values of $\boldsymbol{\mu}$ where $\boldsymbol{\mu}$ is a vector of length n. Subsequently, we plot the expected value of $\boldsymbol{\mu}$, i.e. $E[\boldsymbol{\mu}]$ across all 1000 sampled vectors Y by X.

```r
# Gibbs sampler implementation ---------------------------------------------

t = 1
t_max = 1000

Y = df$Y
N = length(Y)
M = matrix(0, nrow = t_max + 1, ncol = N) # t_max + 1 because we also have X_0
var = 0.2

while (t < t_max){

  for (i in 1:N){

    if (i == 1){
      sd = sqrt(1/2 * var)
      mu = 1/2 * (Y[i] + M[t, i + 1])
    } else if (i == N){
      sd = sqrt(2/3 * var)
      mu = 1/3 * (2 * M[t + 1, N - 1] - Y[N - 1] + 2 * Y[N])
    } else {
      sd = sqrt(2/5 * var)
      mu = 2/5 * (M[t + 1, i - 1] - 1/2 * Y[i - 1] + Y[i] + M[t, i + 1])
    }

    M[t+1, i] = rnorm(1, mu, sd)

  }

  t = t + 1
}
```



Representation of dependence between Y and X

We can see very clearly that $E[\boldsymbol{\mu}]$ can catch the true underlying dependence between Y and X because the two lines for $E[\mu]$ and Y in the plot below are very similar.

Removing noise

It seems that we remove noise since the red line for $E[\mu]$ does not vary as much as the line for Y.

2.5 Task 5: Trace plot and Conclusions

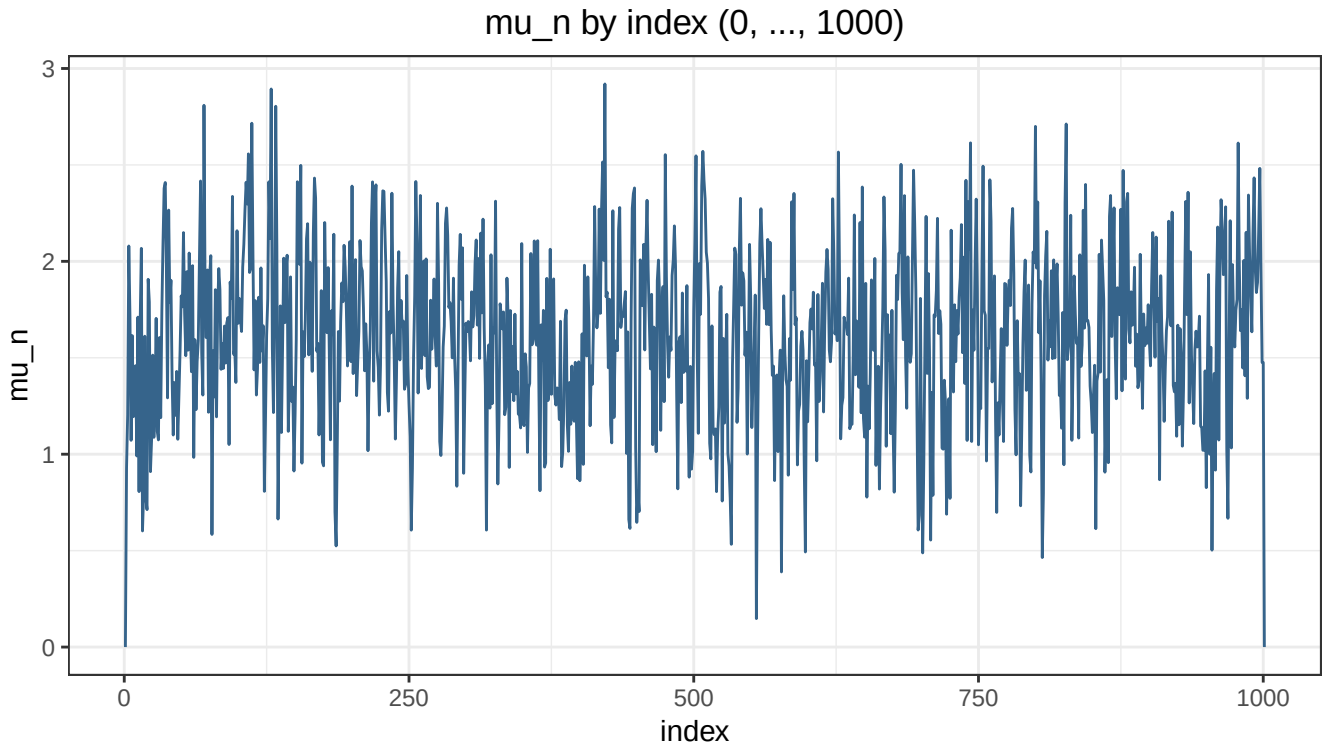
Here, we make a trace plot for μ_n and comment on the burn-in period and convergence.

Burn-in period

It seems like the burn-in period is very short. The initial value μ_0, which was set by us, is 0. The subsequent value is at ca. 0.7 which is already in the range of the other values. Consequently, we can conclude that we have a very short to non-existent burn-in period.

Convergence

Since the burn-in period is very short, we reach the convergence very fast. We can see that the values vary within the same range [0, 3] and are centered at ca. 1.5.



3 Appendix

```r
# Set up general options

knitr::opts_chunk$set(echo = FALSE, message = FALSE, warning = FALSE,
                      cache = TRUE, cache.path = "cache/", fig.path = "cache/")

# options(digits=22)
options(scipen=999)

library(dplyr)
library(ggplot2)
library(magrittr)

set.seed(12345)


# ------------------------------------------------------------------------------
# Assignment 1, Task 1
# ------------------------------------------------------------------------------


# Distribution proportionate to the distribution from which we want to sample
pdf_sampling = function(x) {

  x = ifelse(x <= 0, 0.00001, x)
  res = x^5 * exp(-x)
  return(res)

}

# Proposal distribution LN(Xt, 1)
pdf_proposal = function(x, mu) {

  res = (1 / sqrt(2 * pi)) * (1 / x) * exp(-(log(x) - log(mu))^2 / 2)
  # res = dlnorm(x, meanlog = mu, sdlog = 1) # SAME
  return(res)

}

# Random number generator corresponding to proposal distribution
rng_proposal = function(Xt){

  res = rlnorm(n = 1, meanlog = log(Xt), sdlog = 1)
  return(res)
```

```
}

# Acceptance probability alpha, required to check condition U < alpha(Xt, Y)
alpha = function(Xt, Y){

  num = pdf_sampling(Y) * pdf_proposal(Xt, Y)
  den = pdf_sampling(Xt) * pdf_proposal(Y, Xt)

  if (round(den, 8) == 0){    # Case 1: when denominator is almost zero
    return(1)
  } else {
    return(min(1, num / den)) # Case 2: all other cases
  }

}


# # Example to illustrate that they are the same
# set.seed(123)
# mu = rlnorm(n = 1, meanlog = 3, sdlog = 1)
# x = 3
# (1 / sqrt(2 * pi)) * (1 / x) * exp(-(log(x) - mu)^2 / 2) # option 1
# dlnorm(x, meanlog = mu, sdlog = 1) # option 1


metropolis_hastings_sampler = function(init = 50, t_max = 500){

  # Initialize objects and paramters (We want X_0, X_1, ...X_t_max)
  X = numeric(length = (t_max + 1))
  X[1] = init
  t = 1

  # Conduct iterations of Metropolis Hastings
  while (t < t_max){

    U = runif(1)
    Y = rng_proposal(X[t])

    # a = ifelse(pdf_sampling(X[t]) <= 0, 0, alpha(X[t], Y))

    a = alpha(X[t], Y)
    if (U < a){
      X[t + 1] = Y
    } else {
      X[t + 1] = X[t]
    }
```

```r
    t = t + 1
  }

  res = list(t = 1:(t_max + 1), Xt = X)

}


res = metropolis_hastings_sampler(init = 50, t_max = 500)

# df_plot_task1 = data.frame(t = r1, Xt = res$Xt)
df_plot_task1 = data.frame(t = 1:length(res$Xt), Xt = res$Xt)

ggplot(df_plot_task1, aes(x = t, y = Xt)) +
  geom_point(color = "steelblue4", size = 0.5) +
  geom_line(color = "steelblue3") +
  geom_vline(xintercept = 10, color = "orange") +
  labs(title = "Time Series Plot of Xt by t (orange line = end of burn-in)") +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5))


# ------------------------------------------------------------------------------
# Assignment 1, Task 2
# ------------------------------------------------------------------------------


# Proposal distribution LN(Xt, 1)
pdf_proposal = function(x, Xt) {

  res = dchisq(x, df = floor(Xt + 1))
  return(res)

}

# Random number generator corresponding to proposal distribution
rng_proposal = function(Xt){

  res = rchisq(n = 1, df = floor(Xt + 1))
  return(res)

}


res = metropolis_hastings_sampler(init = 50, t_max = 500)
df_plot_task2 = data.frame(t = res$t, Xt = res$Xt)
```

```r
ggplot(df_plot_task2, aes(x = t, y = Xt)) +
  geom_point(color = "steelblue4", size = 0.5) +
  geom_line(color = "steelblue3") +
  geom_vline(xintercept = 250, color = "orange") +
  labs(title = "Time Series Plot of Xt by t (orange line = end of burn-in)") +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5))


# ------------------------------------------------------------------------------
# Assignment 1, Task 4
# ------------------------------------------------------------------------------


X_0 = 1:10
t_max = 1000
M = matrix(nrow = length(X_0), ncol = t_max)

for (i in 1:10){

  res = metropolis_hastings_sampler(init = i, t_max = t_max)
  M[i, ] = res$Xt[-1] # we remove the initial value

}


n = ncol(M)
k = nrow(M)
v_bar = mean(M)

s_2 = colSums((M - rowMeans(M))^2 / (n - 1)) # vector of length k (with index i)
B = n / (k - 1) * sum((rowMeans(M) - v_bar)^2) # scalar
W = sum(s_2 / k) # scalar

var_hat = (n - 1) / n * W + (1 / n) * B
gelman_rubin_factor = sqrt(var_hat / W)

cat("Gelman Rubin Factor: ", gelman_rubin_factor, "\n")

# Convert M to mcmc.list (unfortunately errors with apply)
l = list()
for (i in 1:nrow(M)){
  l[[i]] = coda::as.mcmc(M[i, ])
}
mcmc.l = coda::as.mcmc.list(l)

# Obtain Gelman and Rubin's convergence diagnostic
```

```r
res = coda::gelman.diag(mcmc.l)
res$psrf


# ------------------------------------------------------------------------------
# Assignment 1, Task 5
# ------------------------------------------------------------------------------


cat("Sample mean of MCMC sequence (task 1):",
    mean(df_plot_task1$Xt[-c(1:10)]), "\n")

cat("Sample mean of MCMC sequence (task 2):",
    mean(df_plot_task2$Xt[-c(1:250)]), "\n")


# ------------------------------------------------------------------------------
# Assignment 2, Task 1
# ------------------------------------------------------------------------------

load("chemical.RData")
df = data.frame(X = X, Y = Y)

# Scatterplot of Y by X [Original Data] ----------------------------------------

ggplot(df, aes(X, Y)) + geom_point(aes(color = "data")) +
  labs(title = "Scatterplot of Y by X [Original Data]") +
  stat_function(fun = function(x) 1.1 * x^(1/4) - 1.1,
                aes(color = "f(x) = 1.1 * x^(1/4) - 1.1")) +
  xlim(0, 50) + ylim(0, 2) +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5)) +
  scale_color_manual("Legend", values = c("steelblue4", "orange"))



# Transforming Y so that relationship between X and Y is linear
df$Y = ((Y + 1.1)/(1.1))^4


# Scatterplot of Y by X [After Transformation] ---------------------------------

ggplot(df, aes(X, Y)) + geom_point(aes(color = "data")) +
  labs(title = "Scatterplot of Y by X [After Transformation]") +
  stat_function(fun = function(x) x,
                aes(color = "f(x) = x")) +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5)) +
```

```r
  scale_color_manual("Legend", values = c("steelblue4", "orange"))


# Reversing data transformation ------------------------------------------------
load("chemical.RData")
df = data.frame(X = X, Y = Y)


knitr::include_graphics("images/marginal_dist_1.jpg")


knitr::include_graphics("images/marginal_dist_2.jpg")


knitr::include_graphics("images/marginal_dist_3.jpg")


# ------------------------------------------------------------------------------
# Assignment 2, Task 4
# ------------------------------------------------------------------------------


# Gibbs sampler implementation -------------------------------------------------

t = 1
t_max = 1000

Y = df$Y
N = length(Y)
M = matrix(0, nrow = t_max + 1, ncol = N) # t_max + 1 because we also have X_0
var = 0.2

while (t < t_max){

  for (i in 1:N){

    if (i == 1){
      sd = sqrt(1/2 * var)
      mu = 1/2 * (Y[i] + M[t, i + 1])
    } else if (i == N){
      sd = sqrt(2/3 * var)
      mu = 1/3 * (2 * M[t + 1, N - 1] - Y[N - 1] + 2 * Y[N])
    } else {
      sd = sqrt(2/5 * var)
      mu = 2/5 * (M[t + 1, i - 1] - 1/2 * Y[i - 1] + Y[i] + M[t, i + 1])
    }
```

```r
    M[t+1, i] = rnorm(1, mu, sd)

  }

  t = t + 1
}


mu = colMeans(M) # Expected mu

df$mu = mu
ggplot(df, aes(x = X)) +
  geom_line(aes(y = Y, color = "Y")) +
  geom_line(aes(y = mu, color = "E[mu]")) +
  labs(title = "E[mu] and Y by X", y = "E[mu] and Y") +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5))


# ------------------------------------------------------------------------------
# Assignment 2, Task 5
# ------------------------------------------------------------------------------

df_plot = data.frame(mu_n = M[, N], index = 1:1001)

ggplot(df_plot, aes(x = index, y = mu_n)) + geom_line(color = "steelblue4") +
  labs(title = "mu_n by index (0, ..., 1000)") +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5))
```