Lab 4

Stefano Toffol (steto820), Mim Kemal Tekin (mimte666)

21 February, 2019

Question 1

We are initially asked to generate samples from the random variable with the following density distribution:

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

We are asked to adopt the Metropolis-Hastings algorithm in order to create the samples and to use the *log-normal* distribution as the proposal distribution X. The density of this proposal distribution is the following:

$$X = e^N = \frac{e^{\frac{-(\ln(x)-\mu)}{2\sigma^2}}}{x\sqrt{2\pi}\sigma} \quad \Longrightarrow \quad \ln(X) \sim N(\mu, \sigma)$$

In `R` this density is computed through the function `dlnorm(·)` and requires the parameters of the associated normal distribution in input. We will now test the Metropolis-Hastings with 4 different starting points and keeping track of the rejected values using the following code:

```r
# Make a function for the terget density
target_density <- function(x) {

  return( exp(-x)*x^5/120 )

}

# The *lnorm() functions from a require the meanlog as argument: in other words
# They request the mu parameter of the normal function --> They require the log
# of the generated values in order to properly work!

# Density of the log-normal function adjusted for the meanlog argument
my_dlnorm <- function(x, mu, sdlog) {

  return( dlnorm(x, log(mu), sdlog) )

}

# Random generation of the log-normal function adjusted for the meanlog argument
my_rlnorm <- function(n, mu, sdlog) {

  return( rlnorm(n, log(mu), sdlog) )

}

# The implementation of the actual algorithm. The three dots are the extra arguments
# that may be requested for the proposal distribution
Metropolis_Hastings <- function(x0, t, proposal, random_proposal, target, ...) {

  # Function for the acceptance ratio (for the if condition)
  alpha <- function(x, x_prev, ...) {

    target_ratio <- target(x)/target(x_prev)
```

```
    proposal_ratio <- proposal(x_prev, x, ...)/proposal(x, x_prev, ...)
    return( min(1, target_ratio*proposal_ratio) )

  }

  # x will be the previously generated number of the sequence
  x <- x0
  # Here we will store the results of the algorithm
  generated <- rep(NA, t)
  # Here we will check whether or not the t-th value has been accepted
  rejected <- rep(0, t)

  # Every iteration will be a generated number
  for(tic in 1:t) {

    # The newly generated number from the proposal
    x_new <- random_proposal(1, x, ...)
    u <- runif(1)

    # Is the value worth to keep if compared with the previous?
    if(u < alpha(x_new, x, ...))  {
      # Yes --> Store it in the results and update the previous value
      generated[tic] <- x_new
      x <- generated[tic]
    }
    # No --> Keep the old value as generated. No need to update x.
    else  {
      generated[tic] <- x
      # The value has also been rejected! Take note of that.
      rejected[tic] <- 1
    }

  }

  return( list(sample = generated, rejection = rejected) )


}

# Set the seed for reprocudibility purposes
set.seed(888)
# Test various starting points
start <- c(1, 25, 50, 500)
# Numbers of generated values
num <- 1e+03
# Generate the samples for every single starting point
ln_samples <- as.data.frame(lapply(start, function(x)
  Metropolis_Hastings(x, num, my_dlnorm, my_rlnorm, target_density, sdlog = 1)))
```

The results of the generation can be observed in Figure 1. For each starting point 1000 samples have been generated. The chains converge in any case, no matter which starting point has been chosen. The values generated fluctuate between 0 and 15 approximately, with frequent strokes consisting of around 5 observations

with the same value in a row. In other words, the algorithm frequently rejects the generated values. In fact the rejection rate is pretty high, close to 60%.
Moreover when the starting point is not inside the convergence range, the generations process requires some iterations before stabilizing. This *burn-in* period is still pretty resticted, consisting of about 20 observations maximum when the starting point is $X_0 = 500$.

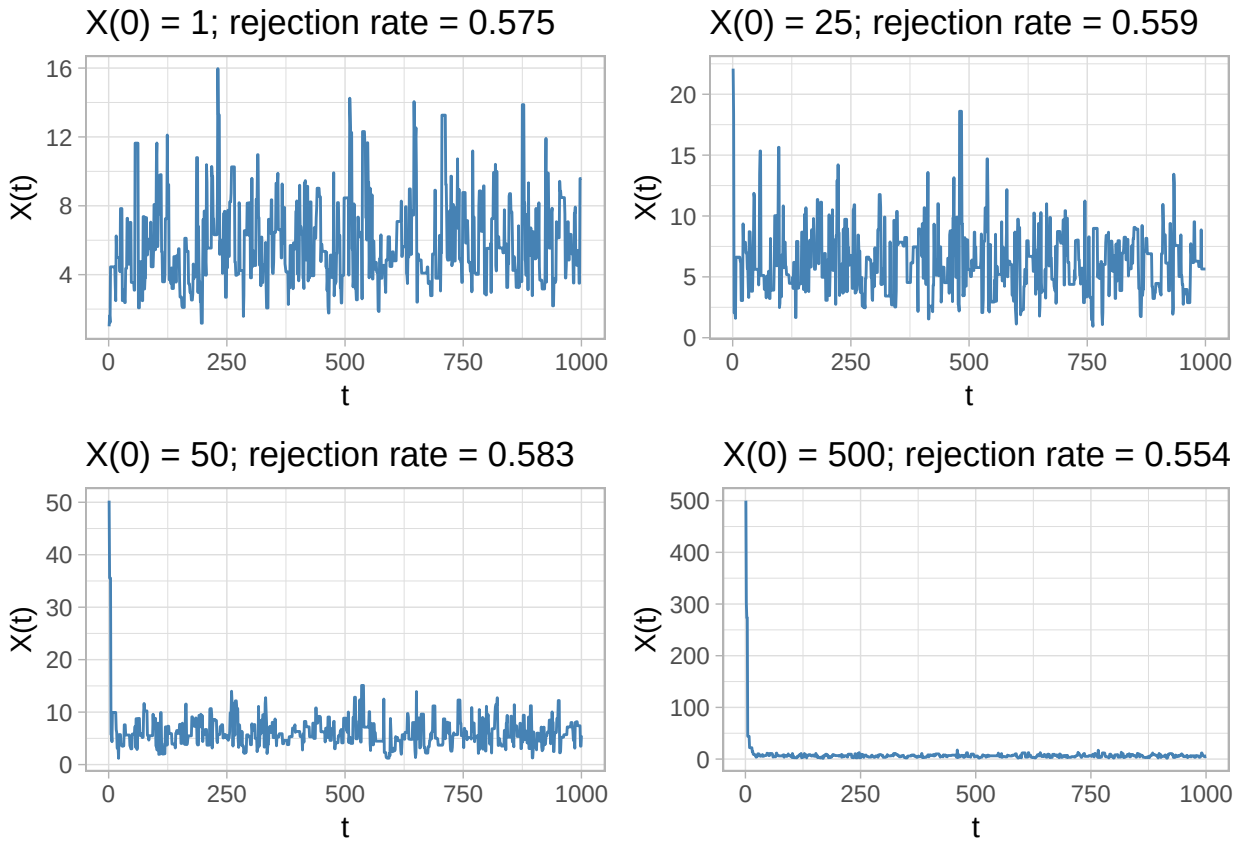


Figure 1: Generated points using a log-normal density as proposal distribution starting from 4 different values.

The successive task requires to generate the same type of data but using a different proposal distribution, a modified chi–square distribution. In order to generate using the chi–square distribution $\chi^2(\lfloor t \rfloor)$, it's necessary to adjust the original `dchisq(·)` and `rchisq(·)` functions from `R`. The rest of the procedure remains the same. The following code has been used to achieve the results:

```
# Adjust the ordinari dchisq(·) function to take just the integer part of a number
dfloor_chi <- function(x, df) {

  return( dchisq(x, floor(df)) )

}

# Adjust the ordinari rchisq(·) function to take just the integer part of a number
rfloor_chi <- function(n, df) {
```

```
  return( rchisq(n, floor(df)) )

}

# New startingg point for chisq: the chain converges much faster
start <- c(1, 25, 50, 500)
# Generate the samples of unchanged size for every single starting point
chisq_samples <- as.data.frame(lapply(start, function(x)
  Metropolis_Hastings(x, num, dfloor_chi, rfloor_chi, target_density)))
```

As we can observe from the plots in Figure 2, the results using this other proposal distribution are still satisfying, but present some differences compared to the previous generation. In fact, this time the number of rejection is less than before (rejecting rate of $\leq 50\%$), and only a few strokes of rejected values are visible in the plots. However, the *burn-in* period is substantially longer, requiring almost 100 observations for high starting point (such as $X_0 = 500$). Nonetheless, the convergence is still reached within a small number of generated values and the fluctuation of the points remains the same. We therefore expect an almost equal precision and accuracy for both method, provided that the *burn-in* observations get removed before hand.

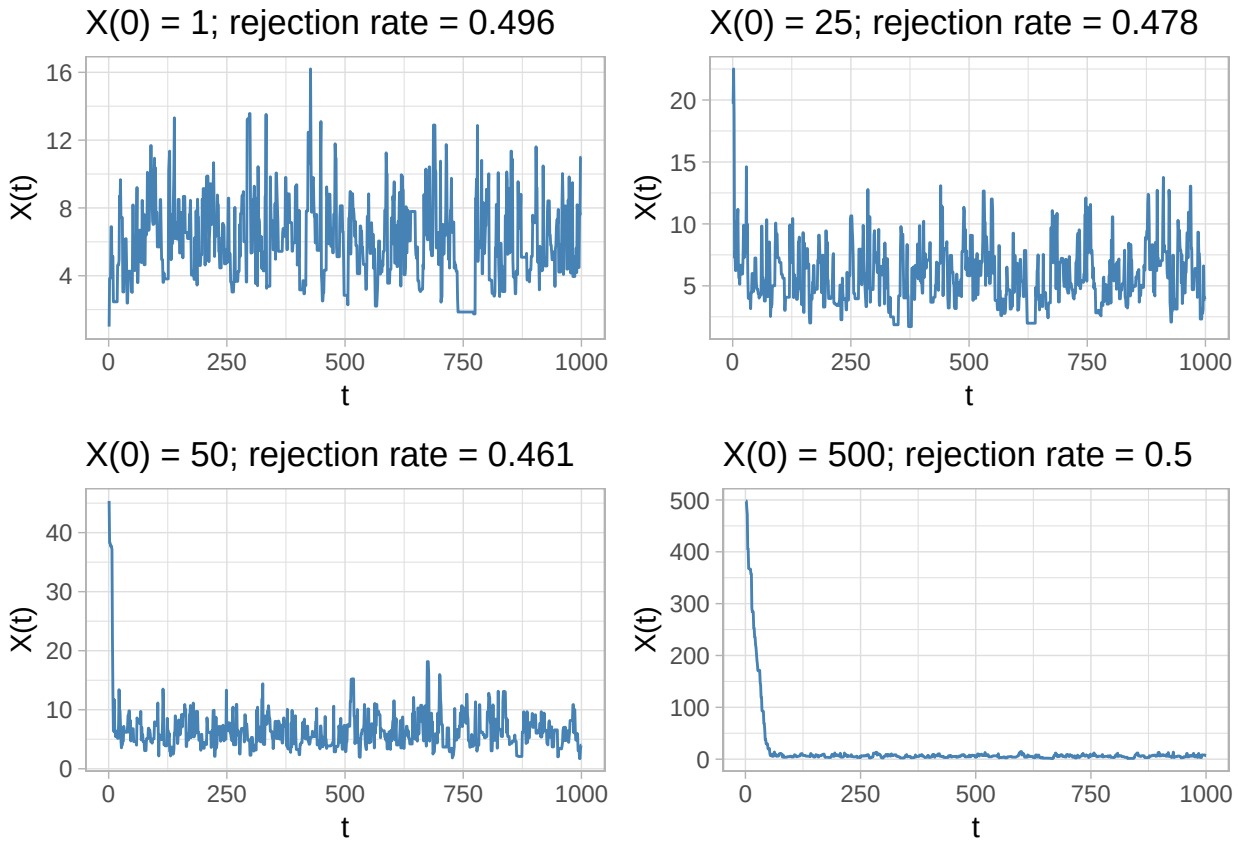


Figure 2: Generated points using a $\chi^2(\lfloor t \rfloor)$ as proposal distribution starting from 4 different values.

If we generate 10 *MCMC* sequences using $\chi^2(\lfloor t \rfloor)$ as proposal distribution with starting points $X_0 = 1, 2, ..., 10$ we can matematically check whether or not the chains actually reach convergence using the *Gelman-Rubin* method. We can exploit the function `gelman.diag(·)` present in the library `coda`. Here is the code used to solve the task:

```
# Load library
library(coda)

# Generate sequences
start_chisq <- 1:10
converg_chisq <- lapply(start_chisq, function(x)
  Metropolis_Hastings(x, num, dfloor_chi, rfloor_chi, target_density))

# Store the results in adeguate object (coda library requires mcmc.list objects)
mcmc_chisq <- mcmc.list()
for(i in start_chisq)
  mcmc_chisq[[i]] <- as.mcmc(converg_chisq[[i]]$sample)

# Compute value of _Gelman-Rubin_ method
gelman_chisq <- gelman.diag(mcmc_chisq)

# BONUS
# Function made to recreate the point estimation
Gelman_Rubin <- function(sequences) {


  # We get rid of the "rejection" argument of the list
  sequences <- lapply(sequences, function(x) unlist(x[names(x)=="sample"]))

  # The name of the variables follow the notation present in slide 21 of lecture 4
  n <- length(sequences[[1]])
  k <- length(sequences)

  means <- sapply(sequences, mean)
  global_mean <- sum(means)/k

  W <- 0
  for(i in 1:k) {
    W <- W + sum((sequences[[i]]-means[i])^2)/(n-1)
  }
  W <- W/k
  B <- (n/(k-1)) * sum((global_mean - means)^2)

  var_v <- W*(n-1)/n + B/n

  return( sqrt(var_v/W) )


}

# Estimation really close to output of function gelman.diag(·): 1.003626
my_gelman_chisq <- Gelman_Rubin(converg_chisq)
```

The output of the *gelman.diag(·)* function returns a point estimaiton of 1.00552 and an upper limit for the CI of 1.01183. For values below 1.2 it can be safe to assume that the chain managed to converge [Brooks and Gelman (1997)]. The result confirm what we observed in the previous graphs.

Knowing that the chains have converged, we can estimate numerically the integral $\int_0^\infty xf(x)dx$ using the samples generated initially. We decided to use data generated from low starting values: for both case the chains start from $x_0 = 1$. The burn-in period (of length 100 approximately) was still discarded when computing the integral.
The function created to achieve the task is the following:

```
mc_integral <- function(generated, alpha = 0.05, burnin = 100) {

  B <- length(generated)
  adj_data <- generated[-c(1:burnin)]
  estimate <- mean(adj_data)
  delta <- qnorm(1-alpha/2) * sd(adj_data) / B
  IC <- c(estimate-delta, estimate+delta)

  return( list(estimate = estimate, IC = IC) )

}

# Estimate integral
value_dl <- mc_integral(ln_samples[[1]][-(1:100)])
value_chisq <- mc_integral(chisq_samples[[1]][-(1:100)])
```

In Table 1 are summarized the results of the computations. The target density however is a *Gamma distribution*. This random variable has a density in the form:

$$f(x;\alpha,\beta) = \frac{\beta^\alpha x^{\alpha-1}e^{-\beta x}}{\Gamma(\alpha)} \quad \text{where, in our case} \quad \begin{cases} \alpha = 6 \\ \beta = 1 \\ \Gamma(\alpha) = 120 \end{cases}$$

An analytical solution of the integral (its mean) is therefore available. In formulas, it is equal to α/β, so in our case the value of the integral is exactly 6.

Table 1: Integral estimations according to the two different proposal distributions.

Proposal	CI - Lower	Point estimate	CI - Upper
Log-normal	6.081067	6.086446	6.091825
Floor chisquare	6.069192	6.074583	6.079973

In conclusion the computed averages of our chains both get close to the real value (at best, overestimating of 0.7 approximately), however their confidence intervals (CI) at level $\alpha = 0.05$ are quite narrow, with width of roughly 0.01, and do not include the real mean.
In our case the $\chi^2(\lfloor t \rfloor)$ distribution is the proposal that leads to the closest results, but this is a consequence of both the seed chosen and, most importantly, of the short length of the chain, which was useful to understand visually the existance of the burn-in period, the trend of the generated samples and the rejection pattern but

is not enough to garauntee a stable estimation. In fact, increasing substantially the number of generated values (from 10^3 to 10^5), the estimates in both case get even closer to 6, with an error of just 0.1 and a width of the CI interval of magnitude 10^{-4} (Table 2). The seed nonetheless still affects the precision of the computations: despite the generally good accuracy of the estimates, when the seed is set to 12345 is the *log-normal* distributions that gets closer to the real value, while the opposite is true when the seed is 54321. Naturally the width of the *CI* is extremely shrinked due to the higher amount of generated data.

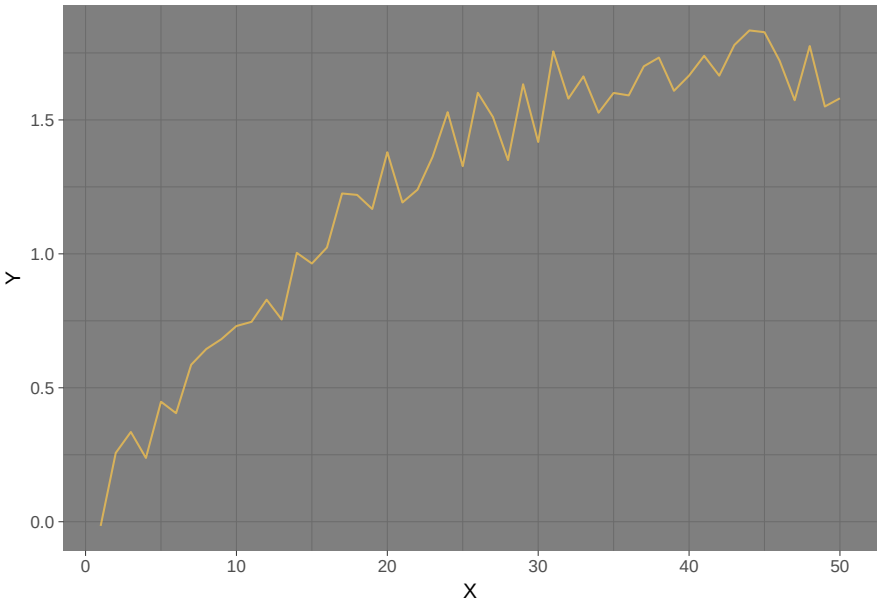
Table 2: Integral estimations according to the two different proposal distributions, using 10^5 values and different seeds.

Seed	Proposal	CI - Lower	Point estimate	CI - Upper
12345	Log-normal	6.010065	6.010114	6.010162
	Floor chisquare	6.012523	6.012571	6.012620
54321	Log-normal	6.035379	6.035427	6.035475
	Floor chisquare	5.994790	5.994838	5.994886

Question 2: Gibbs Sample

Task 2.1

Import the data to R and plot the dependence of Y on X. What kind of model is reasonable to use here?



This data follows an increasing trend in general, but it has serious noise. We can see many fluctuations and instant peaks, deeps. This problem will affect the model badly and the predictions will not be accurate especially after 20, because also the variance of the observations increase with the X. Quadratic linear regression may work for this data, but still this data is not a good data to fit a model.

Task 2.2

A researcher has decided to use the following (random- walk) Bayesian model (n=number of observations, $\vec{\mu} = (\mu_1, ..., \mu_n)$ are unknown parameters):

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

where the prior is

$$P(\mu_1) = 1$$

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

Present the formulae showing the likelihood $p(Y\ \mu)$ and the prior $p(\mu)$. Hint: a chain rule can be used here $p(\vec{\mu}) = p(\mu_1)p(\mu_2|\mu_1)...p(\mu_n|\mu_{n-1})$.

To calculate likelihood formula we need Probability Density Function of Gaussian Distribution which is defined as:

$$\mathcal{N}(\mu,\, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(\vec{y_i} - \vec{\mu_i})^2}{2\sigma^2}}$$

We find likelihood by taking product of PDF as following:

$$P(\vec{Y}|\vec{\mu}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[- \frac{(\vec{y_i} - \vec{\mu_i})^2}{2\sigma^2} \right] = \left(\frac{1}{\sqrt{2\pi\sigma^2}} \right)^n \exp\left[- \frac{1}{2\sigma^2} \sum_{i=1}^{n} (\vec{y_i} - \vec{\mu_i})^2 \right]$$

The prior can be written as:

$$P(\vec{\mu}) = \left(\frac{1}{\sqrt{2\pi\sigma^2}} \right)^n \exp\left[- \frac{1}{2\sigma^2} \sum_{i=1}^{n-1} (\vec{\mu_{i+1}} - \vec{\mu_i})^2 \right]$$

Task 2.3

Use Bayes' Theorem to get the posterior up to a constant proportionality, and then fnd out the distributions of $(\mu_i \mu_{-i}, Y)$*, where* $\mu_{-i}i$ *is a vector containing all* μ *values except of* μ_i

Hint A:

consider for separate formulae for $(\mu_1|\mu_{-1}, Y), (\mu_n|\mu_{-n}, Y)$ *and then a formula for all remaining* $((\mu_i|\mu_{-i}, Y), Y)$.

Hint B:

$$exp\Big(- \frac{1}{d}\Big((x-a)^2 + (x-b)^2\Big)\Big) \propto exp\Big(- \frac{\Big(x - (a+b)/2\Big)^2}{d/2}\Big)$$

Hint C:

$$exp\Big(- \frac{1}{d}\big((x-a)^2 + (x-b)^2 + (x-c)^2\big)\Big) \propto exp\Big(- \frac{\big(x - (a+b+c)/3\big)^2}{d/3}\Big)$$

We have calculated likelihood and prior in the task 2.2 and we can find the posterior by using Bayes' Theorem as following:

$$P(\vec{\mu}|\vec{Y}) \propto P(\vec{Y}|\vec{\mu}) P(\vec{\mu})$$

If we ignore constants in the formulas of likelihood and prior we obtain the formula below for the posterior distribution in general.

$$P(\vec{\mu}|\vec{Y}) \propto \exp\left[- \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - \mu_i)^2 \right] \exp\left[- \frac{1}{2\sigma^2} \sum_{i=1}^{n-1} (\mu_{i+1} - \mu_i)^2 \right]$$

$$P(\vec{\mu}|\vec{Y}) \propto \exp\left[-\frac{1}{2\sigma^2} \left[\sum_{i=1}^{n} (y_i - \mu_i)^2 + \sum_{i=1}^{n-1} (\mu_{i+1} - \mu_i)^2 \right] \right] \propto \exp\left[-\frac{1}{2\sigma^2} \left[\sum_{i=1}^{n-1} \left[(\mu_i - \mu_{i+1})^2 + (\mu_i - y_i)^2 \right] a + (\mu_n - y_n)^2 \right] \right.$$

If we extend for $1, i-1, i, n-1, n$ values of $\sum_i$ the last formula above we can get this:

$$\Big[(\mu_1-\mu_2)^2+(\mu_1-y_1)^2+...+(\mu_{i-1}-\mu_i)^2+(\mu_{i-1}-y_{i-1})^2+(\mu_i-\mu_{i+1})^2+(\mu_i-y_i)^2+...+(\mu_{n-1}-\mu_n)^2+(\mu_{n-1}-y_{n-1})^2+(\mu_n-y_n)^2\Big]$$

Now we can use this posterior distribution to get our sample by finding all μ_i for possible i elements. But we cannot associate this posterior function with any other known probability density function. This means we cannot find directly calculate μ of this distribution without making similar this to another distribution, so we should do some math to obtain known distribution. In sample we have indexes between 1 and n and our sampling target is finding $\mu_{1,...,n}$ values. If we think the marginals of the likelihood we should find $P(\mu_1|\vec{\mu}_{-1}, \vec{Y})$, $P(\mu_2|\vec{\mu}_{-2}, \vec{Y})$, ..., $P(\mu_n|\vec{\mu}_{-n}, \vec{Y})$. But when we think this on the extended to get a generalized formula for ith element we can see that all μ_i that we have in the formula also depend μ_{i-1} and μ_{i+1}. This operation is not possible to perform for first and last elements, so we should create special cases for μ_1 and μ_n elements. We should get only related marginals with our variable from this extended sum formula and the rest of all elements will be a constant for us.

i=1 (first)

When $i = 1$ we can write the first elements of sum operations in the posterior formula.

$$P(\mu_1|\vec{\mu}_{-1}, \vec{Y}) \propto \exp\bigg[-\frac{1}{2\sigma^2}\Big[(\mu_1-\mu_2)^2+(\mu_2-y_1)^2\Big]\bigg]$$

We can see this result is similar to Hint B and we can use the hint to obtain a normal distribution posterior. In this case, we will assume $d = 2\sigma^2$, $x = \mu_1$, $a = \mu_2$ and $b = y_1$. By using these variables we can get this:

$$P(\mu_1|\vec{\mu}_{-1}, \vec{Y}) \propto \exp\bigg[-\frac{1}{\sigma^2}\Big[\mu_1-\frac{\mu_2+y_1}{2}\Big]^2\bigg] \sim \mathcal{N}(\frac{\mu_2+y_1}{2}, \frac{\sigma^2}{2})$$

i=i (general)

We will get the posterior for all μ_i elements. We discussed that we will take all operations which contain μ_i, because in this case the variable is μ_i and all the other μ elements are constants. We extend the sum operation in the likelihood in order to see which paranthesis squares contain μ_i. We can write only them and we can count all other elements as a constant outside. Below we can see the likelihood distribution for ith element:

$$P(\mu_i|\vec{\mu}_{-i}, \vec{Y}) \propto \exp\bigg[-\frac{1}{2\sigma^2}\Big[(\mu_{i-1}-\mu_i)^2+(\mu_i-\mu_{i+1})^2+(\mu_i-y_1)^2\Big]\bigg]$$

This formula is not similar any known distribution again. We can see that this formula has 3 square some which depends on a variable just like Hint C and we can use Hint C in this case. We can assume that $d = 2\sigma^2$, $x = \mu_1$, $a = \mu_{i-1}$, $b = \mu_{i+1}$ and $c = y_1$. By using these variables we can construct this distribution:

$$P(\mu_i|\vec{\mu}_{-i}, \vec{Y}) \propto \exp\bigg[-\frac{3}{2\sigma^2}\Big[\mu_i-\frac{\mu_{i-1}+\mu_{i+1}+y_i}{3}\Big]^2\bigg] \sim \mathcal{N}(\frac{\mu_{i-1}+\mu_{i+1}+y_i}{3}, \frac{\sigma^2}{3})$$

i=n (last)

When $i = n$ we do not have the next element, because n is the last element. So we will do same procedure as before, we will get paranthesis squares which only contain μ_n. The formula is below:

$$P(\mu_n|\vec{\mu}_{-n}, \vec{Y}) \propto \exp\left[-\frac{1}{2\sigma^2}\left[(\mu_{n-1} - \mu_n)^2 + (\mu_n - y_n)^2\right]\right]$$

This formula is not similar any known distribution again. This formula is similar to Hint B again and we can transform tihs distribution to a normal distribution by using Hint B. If we assume that $d = 2\sigma^2$, $x = \mu_n$, $a = \mu_{n-1}$ and $b = y_n$. We can constuct the distribution by using these variables:

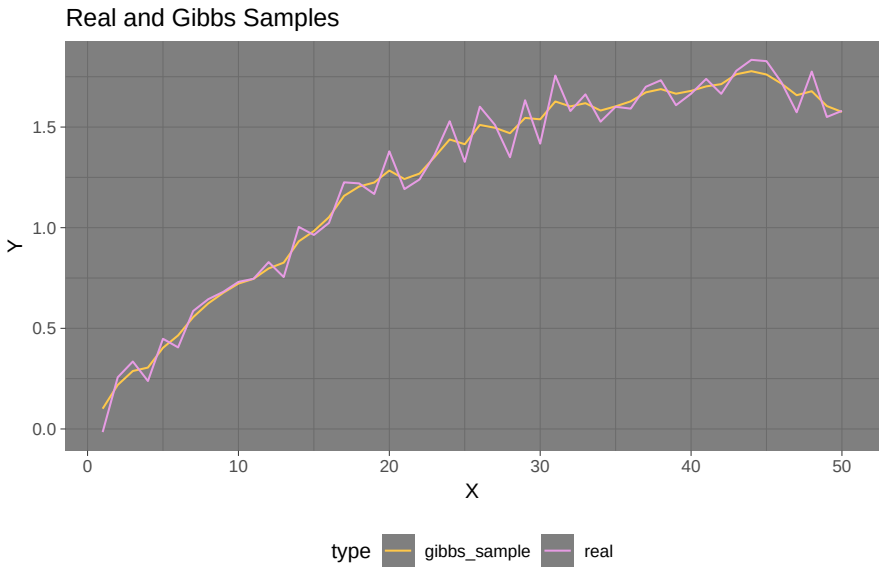
$$P(\mu_n|\vec{\mu}_{-n}, \vec{Y}) \propto \exp\left[-\frac{1}{\sigma^2}\left[\mu_n - \frac{\mu_{n-1} + y_n}{2}\right]^2\right] \sim \mathcal{N}(\frac{\mu_{n-1} + y_n}{2}, \frac{\sigma^2}{2})$$

Now we have the posterior for 3 case and we can implement the sampler in next step. We can see the final likelihood formulas as following:

$$P(\mu_i|\vec{\mu}_{-i}, \vec{Y}) \sim \begin{cases} \mathcal{N}(\frac{\mu_2+y_1}{2}, \frac{\sigma^2}{2}), & i = 1 \\ \mathcal{N}(\frac{\mu_{n-1}+y_n}{2}, \frac{\sigma^2}{2}), & i = n \\ \mathcal{N}(\frac{\mu_{i-1}+\mu_{i+1}+y_i}{3}, \frac{\sigma^2}{3}), & \text{otherwise} \end{cases}$$

Task 2.4

Use the distributions derived in Step 3 to implement a Gibbs sampler that uses $\mu^{-0} = (0, ..., 0)$ as a starting point. Run the Gibbs sampler to obtain 1000 values of $\vec{\mu}$ and then compute the expected value of $\vec{\mu}$ by using a Monte Carlo approach. Plot the expected value of $\vec{\mu}$ versus X and Y versus X in the same graph. Does it seem that you have managed to remove the noise? Does it seem that the expected value of $\vec{\mu}$ can catch the true underlying dependence between Y and X?

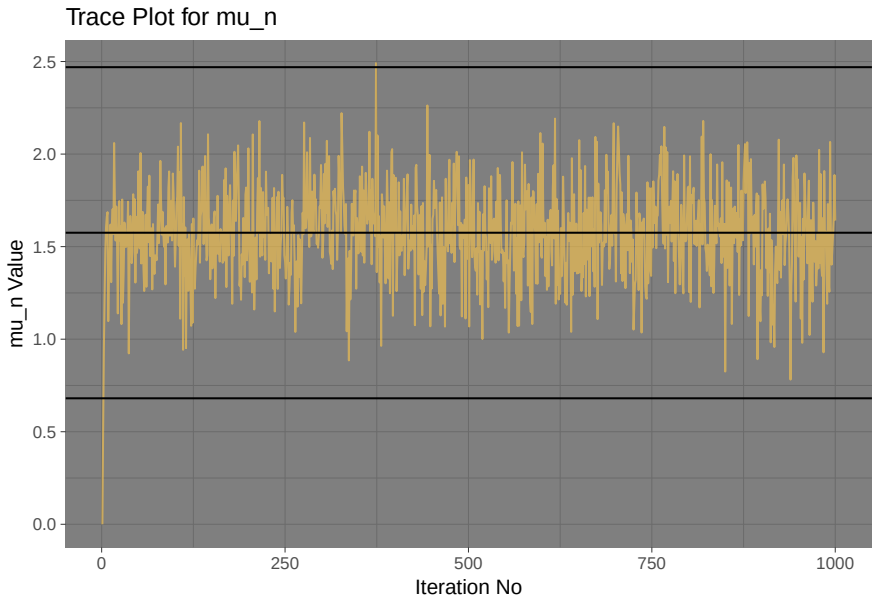


In this task we created a Gibbs Sampler by using the distributions that we obtain from posterior distribution. We run Gibbs Algorithm with 1000 sample for each element in the original data. We have 50 observation in the original data and after gettin 1000 sample for each one, we get their mean.

As we can see from the plot, the original data has much fluctuations and it wasn't proper to fit a model. The data has less noise than after twentieth observation and this noise increases with X. But the sample that we obtain with Gibbs Sampler, we can say that we remove this noise with a significant level. Additionally in general this sample captured the dependence between X and Y. Except some short peaks and deeps it can give many information about trends also.

Task 2.5

Make a trace plot for μ_n and comment on the burn–in period and convergence.



In this plot we can see burn-in period and convergence for μ_n. We can see that the burn-in period is short for this chain. Because it gets stable state in a short period and after getting stable it also keeps to be stationary around mean of μ_n samples and 2*variance range.

Appendix

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


# ------------------------------------------------------------------------------
# A1
# ------------------------------------------------------------------------------


# Make a function for the terget density
target_density <- function(x) {

  return( exp(-x)*x^5/120 )

}

# The *lnorm() functions from a require the meanlog as argument: in other words
# They request the mu parameter of the normal function --> They require the log
# of the generated values in order to properly work!

# Density of the log-normal function adjusted for the meanlog argument
my_dlnorm <- function(x, mu, sdlog) {

  return( dlnorm(x, log(mu), sdlog) )

}

# Random generation of the log-normal function adjusted for the meanlog argument
my_rlnorm <- function(n, mu, sdlog) {

  return( rlnorm(n, log(mu), sdlog) )

}

# The implementation of the actual algorithm. The three dots are the extra arguments
# that may be requested for the proposal distribution
Metropolis_Hastings <- function(x0, t, proposal, random_proposal, target, ...) {

  # Function for the acceptance ratio (for the if condition)
  alpha <- function(x, x_prev, ...) {

    target_ratio <- target(x)/target(x_prev)
    proposal_ratio <- proposal(x_prev, x, ...)/proposal(x, x_prev, ...)
    return( min(1, target_ratio*proposal_ratio) )

  }

  # x will be the previously generated number of the sequence
  x <- x0
  # Here we will store the results of the algorithm
  generated <- rep(NA, t)
```

```r
  # Here we will check whether or not the t-th value has been accepted
  rejected <- rep(0, t)

  # Every iteration will be a generated number
  for(tic in 1:t) {

    # The newly generated number from the proposal
    x_new <- random_proposal(1, x, ...)
    u <- runif(1)

    # Is the value worth to keep if compared with the previous?
    if(u < alpha(x_new, x, ...))  {
      # Yes --> Store it in the results and update the previous value
      generated[tic] <- x_new
      x <- generated[tic]
    }
    # No --> Keep the old value as generated. No need to update x.
    else  {
      generated[tic] <- x
      # The value has also been rejected! Take note of that.
      rejected[tic] <- 1
    }

  }

  return( list(sample = generated, rejection = rejected) )


}

# Set the seed for reprocudibility purposes
set.seed(888)
# Test various starting points
start <- c(1, 25, 50, 500)
# Numbers of generated values
num <- 1e+03
# Generate the samples for every single starting point
ln_samples <- as.data.frame(lapply(start, function(x)
  Metropolis_Hastings(x, num, my_dlnorm, my_rlnorm, target_density, sdlog = 1)))


library(ggplot2)
library(gridExtra)

ln_samples <- ln_samples[,c(1,3,5,7,2,4,6,8)]
names_df <- c("v1", "v2", "v3", "v4", "r1", "r2", "r3", "r4")
names(ln_samples) <- names_df
ln_samples$x <- 1:num


for(i in 1:length(start)) {

  assign(paste("p_ln", i, sep = ""),
```

```
        ggplot(ln_samples, aes_string(x = "x", y = names_df[i])) +
          geom_line(col = "steelblue") +
          labs(x = "t", y = "X(t)",
               title = paste("X(0) = ", start[i], "; rejection rate = ",
                             sum(ln_samples[,i+4])/num, sep = "")) +
          theme_light())

}

grid.arrange(p_ln1, p_ln2, p_ln3, p_ln4, ncol = 2)


# Adjust the ordinari dchisq(·) function to take just the integer part of a number
dfloor_chi <- function(x, df) {

  return( dchisq(x, floor(df)) )

}

# Adjust the ordinari rchisq(·) function to take just the integer part of a number
rfloor_chi <- function(n, df) {

  return( rchisq(n, floor(df)) )

}

# New startingg point for chisq: the chain converges much faster
start <- c(1, 25, 50, 500)
# Generate the samples of unchanged size for every single starting point
chisq_samples <- as.data.frame(lapply(start, function(x)
  Metropolis_Hastings(x, num, dfloor_chi, rfloor_chi, target_density)))


library(ggplot2)
library(gridExtra)

chisq_samples <- chisq_samples[,c(1,3,5,7,2,4,6,8)]
names_df <- c("v1", "v2", "v3", "v4", "r1", "r2", "r3", "r4")
names(chisq_samples) <- names_df
chisq_samples$x <- 1:num

for(i in 1:length(start)) {

  assign(paste("p_chisq", i, sep = ""),
        ggplot(chisq_samples, aes_string(x = "x", y = names_df[i])) +
          geom_line(col = "steelblue") +
          labs(x = "t", y = "X(t)",
               title = paste("X(0) = ", start[i], "; rejection rate = ",
                             sum(chisq_samples[,i+4])/num, sep = "")) +
          theme_light())

}
```

```r
grid.arrange(p_chisq1, p_chisq2, p_chisq3, p_chisq4, ncol = 2)


# Load library
library(coda)

# Generate sequences
start_chisq <- 1:10
converg_chisq <- lapply(start_chisq, function(x)
  Metropolis_Hastings(x, num, dfloor_chi, rfloor_chi, target_density))

# Store the results in adeguate object (coda library requires mcmc.list objects)
mcmc_chisq <- mcmc.list()
for(i in start_chisq)
  mcmc_chisq[[i]] <- as.mcmc(converg_chisq[[i]]$sample)

# Compute value of _Gelman-Rubin_ method
gelman_chisq <- gelman.diag(mcmc_chisq)

# BONUS
# Function made to recreate the point estimation
Gelman_Rubin <- function(sequences) {


  # We get rid of the "rejection" argument of the list
  sequences <- lapply(sequences, function(x) unlist(x[names(x)=="sample"]))

  # The name of the variables follow the notation present in slide 21 of lecture 4
  n <- length(sequences[[1]])
  k <- length(sequences)

  means <- sapply(sequences, mean)
  global_mean <- sum(means)/k

  W <- 0
  for(i in 1:k) {
    W <- W + sum((sequences[[i]]-means[i])^2)/(n-1)
  }
  W <- W/k
  B <- (n/(k-1)) * sum((global_mean - means)^2)

  var_v <- W*(n-1)/n + B/n

  return( sqrt(var_v/W) )


}

# Estimation really close to output of function gelman.diag(·): 1.003626
my_gelman_chisq <- Gelman_Rubin(converg_chisq)


mc_integral <- function(generated, alpha = 0.05, burnin = 100) {
```

```r
  B <- length(generated)
  adj_data <- generated[-c(1:burnin)]
  estimate <- mean(adj_data)
  delta <- qnorm(1-alpha/2) * sd(adj_data) / B
  IC <- c(estimate-delta, estimate+delta)

  return( list(estimate = estimate, IC = IC) )

}

# Estimate integral
value_dl <- mc_integral(ln_samples[[1]][-(1:100)])
value_chisq <- mc_integral(chisq_samples[[1]][-(1:100)])


library(kableExtra)

df_table <- data.frame(a = c("Log-normal", "Floor chisquare"),
                       b = c(value_dl$IC[1], value_chisq$IC[1]),
                       c = c(value_dl$estimate, value_chisq$estimate),
                       d = c(value_dl$IC[2], value_chisq$IC[2]))
kable(df_table,
      col.names = c("Proposal", "CI - Lower", "Point estimate", "CI - Upper"),
      format = "latex", booktabs = T, align = "c",
      caption = "Integral estimations according to the two different proposal
                 distributions.") %>%
  column_spec(1, border_right = T) %>%
  kable_styling(latex_options = c("hold_position"))



set.seed(12345)

start <- 1
num <- 1e+05

test1_lognorm <-
  Metropolis_Hastings(start, num, my_dlnorm, my_rlnorm, target_density, sdlog = 1)
test1_chisq <-
  Metropolis_Hastings(start, num, dfloor_chi, rfloor_chi, target_density)

test1_lognorm_est <- mc_integral(test1_lognorm$sample)
test1_chisq_est <- mc_integral(test1_chisq$sample)


set.seed(54321)

test2_lognorm <-
  Metropolis_Hastings(start, num, my_dlnorm, my_rlnorm, target_density, sdlog = 1)
test2_chisq <-
  Metropolis_Hastings(start, num, dfloor_chi, rfloor_chi, target_density)

test2_lognorm_est <- mc_integral(test2_lognorm$sample)
```

```r
test2_chisq_est <- mc_integral(test2_chisq$sample)

df_table <- data.frame(s = c(rep("12345", 2), rep("54321", 2)),
                       a = rep(c("Log-normal", "Floor chisquare"), 2),
                       b = c(test1_lognorm_est$IC[1], test1_chisq_est$IC[1],
                             test2_lognorm_est$IC[1], test2_chisq_est$IC[1]),
                       c = c(test1_lognorm_est$estimate, test1_chisq_est$estimate,
                             test2_lognorm_est$estimate, test2_chisq_est$estimate),
                       d = c(test1_lognorm_est$IC[2], test1_chisq_est$IC[2],
                             test2_lognorm_est$IC[2], test2_chisq_est$IC[2]))

kable(df_table,
      col.names = c("Seed", "Proposal", "CI - Lower", "Point estimate", "CI - Upper"),
      format = "latex", booktabs = T, align = "c",
      caption = "Integral estimations according to the two different proposal
                 distributions, using $10^5$ values and different seeds.") %>%
  column_spec(2, border_right = T) %>%
  collapse_rows(columns = 1) %>%
  kable_styling(latex_options = c("hold_position"))

library(ggplot2)
load("../datasets/chemical.RData")

ggplot() +
  geom_line(aes(x = X, y = Y), color = "#FFC84B", alpha = 0.7) +
  theme_dark()
# the function that creates gibbs samples as normal dist from an initial sample.
# arguments:
#   - nstep <numeric>       : the count of samples will be generated
#   - x0 <vector>           : initial state of the sample
#   - y <vector>            : the vector contains the real values
#   - sample_var <numeric>  : the variance which will be used in samples
gibbs_sample = function(nstep, x0, y, sample_var){
  n = length(y)
  sample_sd = sqrt(sample_var)
  # create the matrix for store all samples
  samples = matrix(0, nrow=nstep, ncol = n)
  # set the initial sample
  samples[1,] = x0
  # iterate for all other samples that will be create
  for(i in 2:nstep){
    curr_sample = numeric(n)
    prev_sample = samples[i-1, ]
    # print((prev_sample[2] + y[1])/2)
    # generate the first element
    curr_sample[1] = rnorm(1, (prev_sample[2] + y[1])/2, sample_sd/2)
    # generate the elements between first and last element
    for(k in 2:(n-1)){
      mu = (prev_sample[k-1] + prev_sample[k+1] + y[k])/3
      curr_sample[k] = rnorm(1, mu, sample_sd/3)
    }
    # generate the last element
    mu = (y[n] + prev_sample[n-1])/2
```

```
    curr_sample[n] = rnorm(1, mu, sample_sd/2)

    samples[i, ] = curr_sample
  }
  return(samples)
}
set.seed(12345)
n = length(Y)
init_state = numeric(n)
variance = 0.2
g_samples = gibbs_sample(1000, init_state, Y, variance)
mean_sample = colMeans(g_samples)
df_plot = data.frame(X = c(X,X),
                     Y = c(Y,mean_sample),
                     type = c(rep("real",n), rep("gibbs_sample",n)))

ggplot(df_plot) +
  geom_line(aes(x = X, y = Y, color = type)) +
  theme_dark() +
  theme(legend.position = "bottom") +
  scale_color_manual(values = c("#FFC84B", "#E79CE4")) +
  labs(title = "Real and Gibbs Samples")
mu_n_samples = g_samples[, n]

ggplot() +
  geom_line(aes(x = 1:length(mu_n_samples), y = mu_n_samples),
            color="#FFC84B", alpha = 0.6) +
  geom_hline(yintercept=mean(mu_n_samples)) +
  geom_hline(yintercept=mean(mu_n_samples)+2*sqrt(variance)) +
  geom_hline(yintercept=mean(mu_n_samples)-2*sqrt(variance)) +
  labs(title = "Trace Plot for mu_n",
       x = "Iteration No", y = "mu_n Value") +
  theme_dark()
```

Bibliography

Brooks, S. P. and Gelman, A. (1997). General methods for monitoring convergence of iterative simulations. 7:329–339.